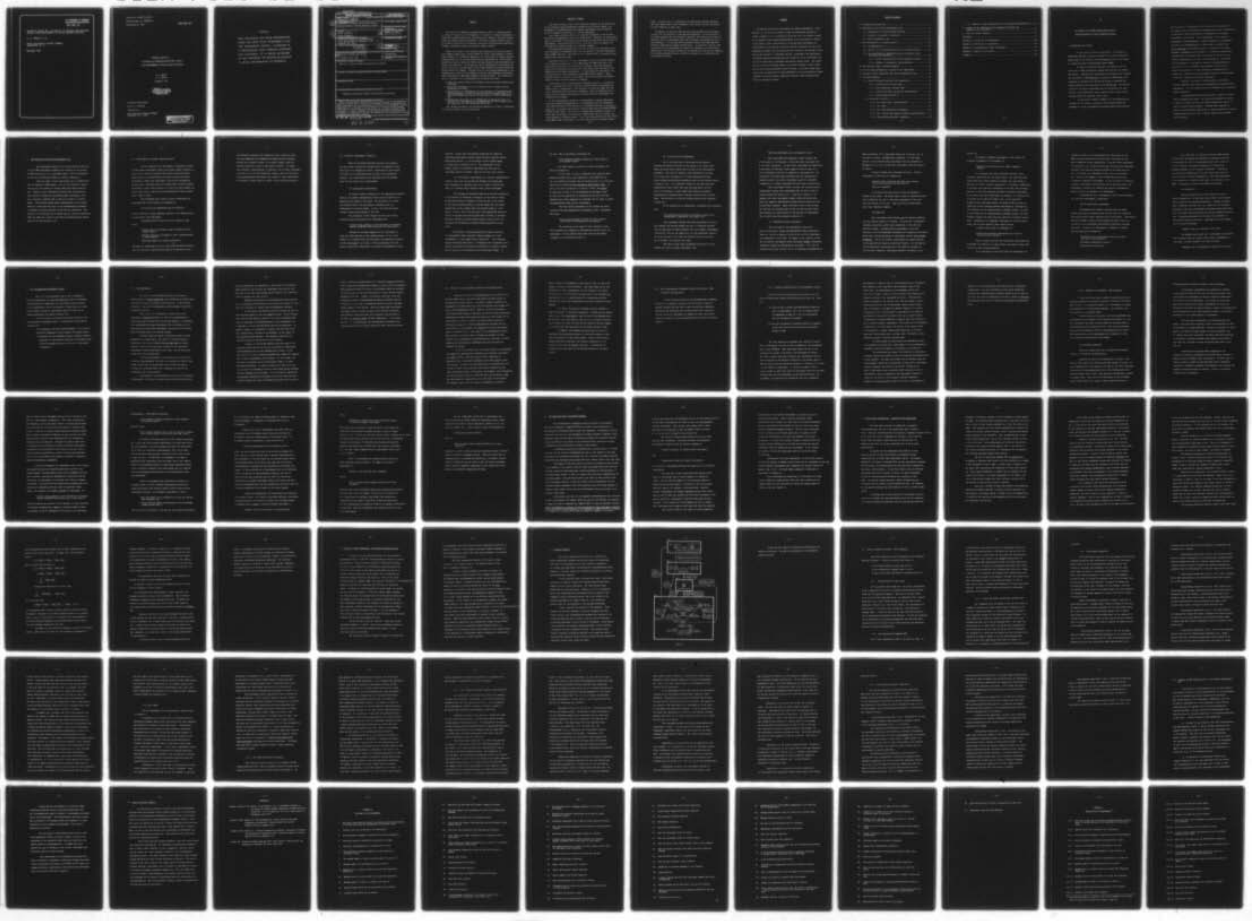
AD-A048 765 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
AN EXERCISE IN SOFTWARE ARCHITECTURAL DESIGN: FROM REQUIREMENTS--ETC(U)
NOV 77 R C ANDREU, S E MADNICK N00039-77-C-0255

UNCLASSIFIED CISR-P010-01-05 NL

1 OF 2
AD
A048 765



TECHNICAL REPORT #3. AN EXERCISE IN SOFTWARE ARCHITECTURAL DESIGN: FROM REQUIREMENTS TO DESIGN PROBLEM STRUCTURE

R. C. ANDREU, ET AL

NAVAL ELECTRONICS SYSTEMS COMMAND
WASHINGTON, D. C.

NOVEMBER 1977

Contract No. N00039-77-C-0255

Internal Report No. P010-01-05

Deliverable No. A004

TECHNICAL REPORT #3

AN EXERCISE IN SOFTWARE ARCHITECTURAL DESIGN: FROM REQUIREMENTS TO DESIGN PROBLEM STRUCTURE.

R. C. Andreu

S. E. Madnick

November, 1977

Principal Investigator:

Prof. S. E. Madnick

Prepared for:

Naval Electronics Systems Command
Washignton, D.C. 20360

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: (9) Technical Report, #3
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): (6) An exercise in software architectural design: From requirements to design problem structure.
5. TYPE OF REPORT & PERIOD COVERED
6. PERFORMING ORG. REPORT NUMBER: P010-01-05

7. AUTHOR(s): (10) Rafael C. Andreu; Stuart E. Madnick
8. CONTRACT OR GRANT NUMBER(s): (15) N00039-77-C-0255

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Information Systems Research, M.I.T. Sloan School of Management - E53 - 330, Cambridge, MA, 02139
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS: (12) 129p.
12. REPORT DATE: (11) Nov[illegible], 1977
13. NUMBER OF PAGES: 121

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (14) CISR-P010-01-05, CISR-TR-3
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software architectural design; design problem structuring

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two of the main activities required for the application of the methodology explored in this project, namely, the assessment of design interdependencies among requirements and the interpretation of subsets of requirements as design subproblems, are investigated in the context of a concrete design problem. Guidelines for carrying out these activities are introduced and their usage illustrated. A design framework for the problem analyzed is identified and discussed; its study points out that the methodology investigated here produces interesting and unforeseen results.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 590

AB

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management; it consists of a group of Management Information Systems specialists, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, generating and maintaining application software, information systems and decision support systems.

Within the context of the research effort sponsored by the Naval Electronics Systems Command under contract N00039-77-C-0255, CISR proposed to conduct basic research on a systematic approach to the early phases of complex software systems design, one of the main goals being the development of a well defined methodology aimed at explicitly filling the gap between system requirements and program specifications. At the heart of such a methodology is the structuring of the initial set of requirements so as to make apparent the design trade-offs existing among its elements. The main focus of the proposed methodology is the decomposition of that set into subsets of strongly interdependent requirements which would define a meaningful framework for system design. The research project is organized so as to investigate the following four areas:

1) Graph-like representation of requirements sets and suitable decomposition techniques,
2) Design and development of a set of software tools to support the set decomposition activity,
3) Identification of a methodology for the assessment of interdependencies among requirements, as well as guidelines for the interpretation of the obtained decompositions and for the coordination of design subproblems, and
4) Experimental application of the methodology and supporting tools to a specific case, with emphasis on recommendations for their practical use and comparison with more traditional approaches.

This document focuses on the activities carried out at CISR to investigate the third and fourth areas outlined above.

EXECUTIVE SUMMARY

This report focuses on two of the activities required for the application of the systematic design methodology explored in this project, namely, the assessment of interdependencies among design requirements and the interpretation of subsets of requirements as design subproblems.

Since these activities require the personal involvement of the designer, they can't be approached in a strictly formal basis. Therefore, we investigated them in the context of an actual design problem, starting with a concrete set of requirements. This allows the discussion to be more lively and permits the illustration of certain concepts with specific examples.

The requirements employed came from a set of about 100 requirements originally prepared by a Government Agency for the purpose of Data Base Management System (DBMS) selection. With this set as a starting point, the present report explores the following issues:

(1)Desirable properties of design requirements: Since the original purpose of the employed requirements was not DBMS design but DBMS selection, a few requirements were found to be irrelevant for design; these were discarded. Analyzing the remaining ones pointed out a few which, given the goals of the methodology, were judged inappropriate. In this process we identified a set of characteristics that the employed requirements should exhibit. These characteristics are outlined in the report; any of the original requirements found to violate them were changed accordingly.

(2) The nature of the interdependency assessment activity: A conceptual model is described which may be employed to assess design interdependencies between pairs of requirements. Our experience in using this model with the set of requirements mentioned above suggested a series of guidelines that we found useful to conduct the assessment activity; these are highlighted and documented with examples.

(3) The interpretation of subsets of requirements as design subproblems: Using the assessed interdependencies, the requirements set was descomposed into subsets by means of the decomposition techniques described in Technical Report #1. An analysis was performed aimed at identifying the design problem defined by each subset. This analysis produced some interesting results. In some cases, collections of requirements traditionally seen as a logical unit were separated in different subsets; their close interactions with other design requirements made it desirable to consider them as parts of separate subpro-

blems. In other cases, our experience with DBMS design problems suggested that some subsets were in fact incomplete; this pointed out that the initial set was incomplete to begin with.

In general, the exercise reported herein has provided us with valuable insights into the potential of the explored methodology. On the one hand, it has become apparent that the designer's expertise and intuition still play an important role; the methodology helps him to organize his thoughts and to be more systematic. On the other hand, unforeseen results were obtained which helped to understand the design problem better. In particular, the fact that the initial set of requirements was found to be incomplete in certain areas implies that a second pass, begining by the completion of that set as suggested by the above mentioned results, should be performed. This is planned as the next research activity.

FOREWORD

The exercise described in this report was largely experimental. Since, moreover, it was the first attempt to apply the methodology proposed in [Andreu & Madnick 77] to a problem of non-trivial size, care should be taken not to generalize the results reported herein in an indiscriminate fashion. In particular, certain characteristics of the employed requirements set found useful for the exercise, as well as specific guidelines for the interdependency assessment activity which were well suited for the problem at hand, may be less relevant in a different setting. Furthermore, the implications of specific decisions regarding both the formulation of the requirements set and the interdependency assessments made warrant further study. This report has been prepared to document some of our current thoughts on these issues and to spark further discussion within our group. As we gain further experience with the application of this methodology to this and additional design problems, we hope to be able to make more definite claims regarding these issues in subsequent reports.

√

TABLE OF CONTENTS

AN EXERCISE IN SOFTWARE ARCHITECTURAL DESIGN: FROM REQUIREMENTS TO DESIGN PROBLEM STRUCTURE.

1. Introduction and Overview

In this paper we report our experience in applying the methodology proposed in [Andreu & Madnick 77] for the architectural design stage of the software system development cycle, to the design of a generalized Data Base Management System (DBMS).

To make the experience realistic, we decided to use an actual set of DBMS requirements prepared some time ago by a Government Agency. Although these requirements were originally put together for the purpose of DBMS selection, as opposed to DBMS design, they constitute a representative case and are useful in exploring the issues involved in the application of the methodology. The characteristics of the initial requirements set are discussed in the next section; a few modifications that we made in order to make it more design-oriented are described there as well.

As described in [Andreu & Madnick 77], the methodology we advocate is, in a sense, an attempt to fill the gap between the statement of system requirements and software module specifications,

with the goals of (i) making the latter explicitly consistent with the former and (ii) structuring the design process by way of identifying a collection of design subproblems as independent of one another as possible. At the heart of that methodology is the concept of requirements' interdependencies. Loosely speaking, two requirements are said to be interdependent if it makes sense to consider them at the same time in the design process. A collection of strongly dependent requirements can be thought of as defining a design subproblem which can be attacked almost in isolation as long as it is relatively independent of the rest of the requirements. The concept of interdependency is discussed in section 3. Interdependencies in the initial set of requirements were assessed in order to give a graph-like structure to it, thus allowing its decomposition in a formal way. Guidelines for the assessment process are also discussed in section 3, drawing on our actual experiences.

Section 4 is devoted to the description of the graph structure obtained, based upon the interdependencies assessed between requirements. Its main characteristics are discussed and a preliminary analysis performed.

The decomposition techniques outlined in [Andreu 77a] were used to decompose that graph. The results are described in section 5, where it is noted that the first graph decomposition results in only a few fairly large subgraphs. Although these subgraphs do point out the main design subproblems, it seems appropriate to further ecompose each of them in order to better organize and understand their structure.

Although doing this further decomposition is likely to produce an overall partition whose evaluation measure is inferior to that of the first decomposition, the main subproblems are so weakly related to one another that it makes sense to consider each of them in isolation. In practice this would mean to organize the design process in a few well defined and almost independent subproblems, the interfaces among them being easy to control due to their simplicity. Each of these "main" subproblems (MS) then can be analyzed and structured independently of the remaining ones. Note that structuring each MS in isolation, although done by using the same decomposition techniques used before, will not, in general, generate the same organization that a decomposition of the more complete initial set would. The reason is that the decomposition techniques employed use interdependencies' information in a way relative to the elements in the set being decomposed, as opposed to one absolute scheme that would "place" each element in some "space" independently of the locations of other elements in that same space.

Section 6 reports the results of the second decomposition step, and in section 7 the obtained design framework is displayed and analyzed. A couple of very simple operations are proposed which permit a meaningful interpretation of the subproblems and their interactions. The implications for design are discussed and traced back to the initial set of requirements.

The set of tools made available through the "decomposition package" described in [Andreu 77b] are used throughout the analysis outlined above. Section 8 is devoted to comments on the usefulness of such a package as it currently stands. Recommendations are made regarding some new facilities which we think could make it more convenient to use.

Section 9, finally, summarizes our experience and points out some topics for further research whose investigation would, in our opinion, improve the usefulness of the proposed methodology.

2. Description of the Initial Requirements' Set

The requirements employed in the study reported herein was a set of DBMS requirements originally put together for the purpose of DBMS selection by a U.S. government agency. The 97 requirements in that set as originally defined are listed in Appendix A.

Since the usage of these requirements was to be DBMS selection (as opposed to DBMS design), a few of them referred to issues that are not too relevant for design; these were removed from the set. Moreover, other requirements presented some characteristics that should be avoided when specifying design requirements; this led to having to rephrase some of them and to merge or to delete others. The rationale behind these transformations of the requirements set, which were done prior to any interdependency assessment or analysis is briefly discussed in the next two subsections. Subsection 2.2 in particular can be looked upon as a series of guidelines which are important to keep in mind when establishing design requirements of the type we wish to use.

2.1. Requirements too biased towards selection

A first scanning of the requirements in Appendix A pointed out that some of them didn't specify any evident design constraint or goal; rather, they were concerned with clearly selection-related issues, which is not surprising, given their original motivation. We decided to remove such requirements from further consideration at the outset. Although, had we not removed them, we would probably have done so later on (see section 4), we felt that their removal was safe at that point in time since no relationship with other requirements seemed to exist.

The requirement most clearly in these circumstances was requirement #67 of the original set (Appendix A):

> "Two hour on call maintenance must be available."

This is obviously an issue completely external to the design process. We thus removed that requirement.

Requirements #83, 84 and 85 were also removed at this point:

> "Dynamic control of working storage requirements within program region."
>
> "Dynamic relocation of programs to avoid 'checkerboarding' of usable memory."
>
> "Additional memory will enhance performance."

Although such requirements are not as obviously selection-oriented as #67, the decision to remove them was made on the ground that they

were probably included in the original set just to make sure that the chosen DBMS had been implemented through advanced techniques, perhaps in an attempt to check, in an indirect manner, that the selected system was a piece of high quality software. Of course, this rationale, being basically a conjecture, can be easily challenged. As pointed out in the next subsection, however, there are other more convincing arguments which support the removal of those requirements; the decision to remove them is a result of this two-fold motivation.

2.2. Desirable requirements' properties

While the preceding discussion focused on requirements that were clearly unrelated to design issues, the emphasis in this subsection is in somewhat more subtle characteristics which should be avoided in the requirements used as input to the methodology employed. Such characteristics are discussed below and examples given from the initial set of requirements.

(a) Implementation independence:

As argued in [Andreu & Madnick 77], the requirements shouldn't indicate any possible implementation scheme that may eventually be employed in the design of the system. The motivation is to avoid any technical bias in the requirements' statements that can predetermine the eventual design. In fact, this has been typical of many software system designs in the past.

The requirement in the original set which most clearly presented these characteristics was requirements #35:

> "Literals being compared to field variables are validated against acceptable field values prior to data search."

Although the strategy suggested by this requirement is sound and indeed employed in many implementations, it is a clear example of a statement which specifies <u>how</u> to do something as opposed to just saying <u>what</u> is to be done. This was precisely one of the points we specially stressed when we proposed the methodology in this

research. In this case, the eventual system must be capable of doing data search given a literal value and this is all the requirements should specify. It is the purpose of later design stages to decide upon a strategy to do it. The concreteness of this example allows us to discuss in some detail why this type of requirement should be avoided. There are at least three reasons:

(i) The explicit consideration of a specific implementation scheme at this stage can preclude the designer from considering other solutions; in principle, there is no reason to believe that there is a universal "best" scheme to meet a given requirement.

(ii) Although the proposed scheme may be the most appropriate in the majority of situations, choosing it at the outset is wrong because the decision of adopting it should be done in the context of the overall design problem; it just may happen that the present design falls in the category where the proposed scheme is a bad choice. In fact, one of the main aims of the methodology advocated here is to get away from such pitfalls while avoiding, if possible, having to consider the entire design problem at once; this is one motivation for attempting to identify design subproblems the way we do.

(iii) Finally, stating implementation schemes along with the requirements almost inevitably implies merging two or more requirements. In the example above, saying that "...are validated against acceptable field value..." presupposes that "acceptable field values" are defined, which should be specified in a separate requirement if they are to be supported by the system under design. In

our case, this is specified by requirement #6:

> "Field definition permits validation of input datum as to acceptable values."

For these reasons, we decided to delete requirement #35 from the original set.

At this point, it must be emphasized that plausible implementation schemes such as that suggested by requirement #35 are not to be absolutely excluded from the design process; it is only that they should be avoided at the requirement specification stage. In fact, as it will be pointed out in section 3 below, they should be taken into account at the interdependencies assessment stage. We will have the opportunity of discussing how precisely the same implementation scheme suggested by requirement #35 is useful to assess interdependencies among certain requirements.

There was another requirement in the original set which touched on the same implementation independence issue. Requirement #36 stated:

> "Data meeting selection criteria (or their pointers) can be used for subsequent query processing."

The inclusions of the words "or their pointers" in the above statement also suggests an implementation scheme to meet the requirement. We thus deleted those words from the requirement statement to be considered from now on.

(b) System structure independence

One of the objectives of the design process being to determine the eventual structure of the system, it is obvious that system structure biases should also be avoided in the requirements' statements. This issue is similar to that discussed in (a) above in the sense that its essence is to avoid influencing the final design through preestablished assumptions about how the eventual system is going to look. It is different, though, in the sense that such assumptions are more global than those discussed above, as they refer to the system as a whole rather than hinging on plausible implementation techniques aimed at meeting certain specific requirements. The motivation to avoid such biases parallels that discussed in point (a).

In the original set of requirements, requirement #29 illustrates this:

> "File maintenance facilities are available (such as file reorgnization, compression, copy, dump, etc.)."

This requirement implies stand alone maintenance utilities. This assumes that whatever schemes are used to reorganize and maintain the data base on normal operation (see, for example, requirements # 86 and 87), they are not appropriate for bulk maintenance purposes. As it was the case in (a), this may actually turn out to be true, but it shouldn't be stated at the outset.

The need for bulk, batch maintenance utilities is in fact already set forth by another requirement, #16:

"Data base maintenance can be performed by batch."

Thus, requirement #16 expresses a "what" condition and in this sense it is acceptable; on the other hand, requirement #29 is its "how" counterpart. In other words: requirement #16 establishes a need which, in principle, doesn't have to be met by means of a set of independent utilities as suggested by requirement #29. The possibility of sharing common processing with the implementation schemes devised to meet other requirements should be considered. Consequently, we deleted requirement #29 from further consideration.

Other requirements found to violate this "system structure independence" property were requirements #83, 84 and 85, which we already deleted (see section 2.1). When read carefully, it is apparent that these requirements assume a system structure under which the suggested implementation schemes would be beneficial from an efficiency viewpoint. Again, this may be true in most systems, but this does not guarantee that it will for the one under design. This completes the justification for the deletion of requirements #83, 94 and 85 which was advanced in section 2.1.

(c) Independence among requirements

This title may be a bit misleading at this point. Since we are going to assess interdependencies among requirements later on, how are we going to be able to do so if the requirements are independent to start with? The answer to this question is that here we advocate independence among requirements without considering alternative appropriate implementation techniques. As it will be discussed in the next section, one way of assessing interdependencies

among requirements is to think about which ones can be met, say, by the same, or similar, implementation techniques. At this stage, however, we are concerned about requirements that are semantically dependent, or, to put it another way, requirements which are implied by others.

A specific example will illustrate the point. Consider requirements 71 and 88 in the original set:

> "Removable disks containing data base can be mounted and processed during an on-line session."
>
> "Disks are removable."

It is obvious that the latter of these requirements is implied by the former. The former states a need for convenience which can be physically met only if the latter requirement is met (note that the converse is not true). Thus, the latter is unnecessary, and it was, therefore, deleted.

(d) Simplicity

Each requirement should address <u>one</u> well-defined capability that the eventual system is to possess. In particular, requirements' statements specifying several such capabilities at the same time should be avoided. Grouping similar requirements is the final objective of the interdependency assessment and decomposition processes, which aim at identifying groups of requirements similar <u>from a design standpoint</u>. Very often, however, requirements are grouped together at the specification stage in an arbitrary fashion. For the purposes of the methodology investigated here, this should be avoided; we will say that "composite" requirements shouldn't be present in the

initial set.

An example of composite requirement in the initial set presented in Appendix A is requirement 14:

> "Records in the data base can be added, changed or deleted."

It is apparent that this requirement specifies three different capabilities for the target system. The reason why these capabilities were stated together is probably that they belong to what may be called a "class of capabilities." In general, capabilities in the same class need not be considered at the same time for design purposes, so that they should be specified as separate requirements. In the case of the above requirement, for example, it is obvious that, if records are of fixed size (which, note, is not known from looking at that requirement alone), the design issues that must be taken into account regarding possible implementation techniques are very different for say adding records or changing records. In order to avoid biases in the interdependency assessment process, the three different capabilities above should be specified in separate requirements. Consequently, we "exploded" requirement 14 in three requirements, one for each capability (add, change, delete).

A similar case is that of requirement 31:

> "Boolean and relational conditionals can be used in record selection criteria."

Again, although "boolean" and "relational" conditionals may be thought of as similar in a logical sense, this doesn't assure that they are so from a design standpoint.

It is interesting to note that those two requirements, as

originally stated, are not particularly well suited even for the DBMS selection purposes that motivated them. The reason is that given a DBMS with certain capabilities, it may meet those requirements only partially (e.g., it may allow boolean criteria but not relational criteria); this poses an evaluation problem in the sense that one can't tell whether the requirement is met or not in an absolute way. In fact, this suggests a rule of thumb which can be used to pinpoint "composite" requirements: If one can think of a situation in which a requirement can be partially met in the above sense, that requirement is likely to be composite and should be decomposed in simpler ones for the purposes of our methodology.

Following this rule, requirements 14 and 31 were decomposed in 3 and 2 new requirements, respectively.

(e) No "stand alone" requirements

Requirements only remotely concerned with the target system should also be avoided because they are likely to complicate the design process without resulting in any clear benefit. Requirements which, for example, specify system capabilities which in fact can be added on to an operational system without difficulty illustrate this point. In the set of requirements of Appendix A, consider the following ones (43 though 46):

> "Procedures can be written which interact with data base and on-line terminals."
>
> "Procedures interactively created."
>
> "Procedures can utilize use

It is clear that, as long as the eventual DBMS provides its users with an adequate data manipulation language (called for by requirement 30), the four requirements above call in fact for a "macro facility" allowing the combination of manipulation language statements in the so-called "procedures." Although we used subjective judgement here, the point can be made that such a macro facility is fairly independent of "true" DBMS requirements in the sense that it could be added easily, in principle, to any DBMS. Consequently, the above requirements were deleted.

(f) Plausability

The final property that we will discuss here can be called "plausability," in the sense that we wish to avoid requirements calling for the physically impossible. Although in general what is impossible and what isn't is a function of the available technology, there may be requirements specifying capabilities which can't be possibly achieved because they violate fundamental physical laws. Accepting requirements of this type would determine an unsuccessful design at the outset. Requirement 79 in the original set is one such requirement:

"Response time not a function of file size."

No designer would accept such a requirement in good faith (one can always make the response time infinity independently of file size). We thus reworded it to read as follows:

"Response time as independent as possible of file size."

Other rewordings such as "average response time less than ... " may be more acceptable to some people, but this is not central to our discussion here.

2.3 Final Requirements Set

Appendix B lists the resulting set of requirements after the initial set of Appendix A was modified as discussed above. There are 89 requirements in all; interdependencies were assessed among these as described in the next section.

3. The Interdependency Assessment Process

Once a set of requirements such as that of Appendix B has been established, the proposed methodology calls for assessing "interdependencies" among them, the idea being to give structure to the requirements' set for the purpose of identifying, in a systematic fashion, subsets of requirements which will give rise to design subproblems (see [Andreu & Madnick 77]).

In this section, we discuss several issues related to the interdependency assessment process. In particular, we pay special attention to:

1) The *meaning* of the word "interdependency" in our context;
2) How the assessment activity supports the design process;
3) Guidelines for the assessment process; different possible approaches to that process, along with illustrations drawn from the assessment process for the set of requirements of Appendix B.

3.1. Interdependencies

The goal of the methodology advocated here being the identification of design subproblems, the interdepencies defined among requirements should be consistent with that goal. In this section we outline a concept of interdependency which attempts to make this consistency explicit.

First of all, it is important to recall that we are interested in design subproblems, so that the interdependencies among requirements should reflect this design emphasis. Accordingly, the interdependencies among requirements will be defined so as to make explicit how different requirements interrelate from the standpoint of meeting them in the eventual design.

Two basic ideas are useful to build the concept of interdependency as outlined above. Our goal is to identify groups of requirements that can be considered at the same time for design purposes. There are two main ways in which one can think about requirements being interrelated in such a way. We call these ways "trade-offs" and "concurrencies."

Two given requirements in the initial set are said to be "trade-off interdependent" (TI) when it is possible to think of some scheme in which the two requirements conflict with one another. In practice, conceptual models can be imagined in which the two requirements are in open conflict.

Alternatively, two requirements are said to be "concurrency interdependent" (CI) when a conceptual model can be devised in which

the two requirements are compatible; in other words, the conceptual model points out ways in which the requirements can be met at the same time in the sense that meeting one of them will, as a byproduct, result in meeting the other as well.

Examples of both types of interdependencies drawn from the set of requirements described above will be given in section 3.3 below; they will contribute to clarify those interdependency concepts.

At this point, the question arises of how does one come up with conceptual models of the type suggested above. We will discuss this issue at some length in section 3.3, by way of examples taken from the assessment of interdependencies in the requirements' set of Appendix B and thus drawing mainly from our own experience. In general, though, it is important to realize that this step is the most creative one on the part of the designer, in the context of the methodology described here. The designer's expertise and intuition can play a central role in this activity.

Finally, it will become apparent below that many of the conceptual models that can be used for the purpose of identifying interdependencies have a clear implementation "flavor," in the sense that it is often a possible implementation scheme that suggests ways in which requirements can interrelate. At first sight, this may seem to contradict one of our main goals, namely, to avoid implementation biases. It should be pointed out that this is not so as long as the assessment process is open minded enough regarding possible implementation techniques. It is the implicit adoption of one concrete implementation technqiue at the outset that has traditionally produced the kind of implementation biases which we wish to

avoid. Generating interdependencies by considering several (and maybe even incompatible) implementation schemes to come up with the conceptual models mentioned before is in fact a step forward toward the avoidance of the design biases that are characteristic of many software systems designed in the past. Finally, it should be relatively clear that assessing the kind of interdependencies among requirements that are of interest to us by way of considering different possible implementation schemes is only natural: Since the established requirements are finally going to be met by means of the available technology, the different alternatives that such technology can offer must be an input, in a generic sense, (without commitments), to the design process. In our methodology, the interdependency assessment step constitutes the point in time in which that input enters the process.

3.2. The Role of the Assessment Activity in the Design Process

Assessing the kind of interdependencies just outlined in an explicit manner fills a gap that traditionally has been present in the design of software systems. As argued in [Andreu & Madnick 77], implementation considerations are often kept implicit and brought into play at the implementation stage. However, the final structuring of the system under design is typically the result of some preconceived implementation scheme which implicitly drives the early stages of the design process so that, in effect, the final system structure is determined a priori by a concrete implementation approach (the one most familiar to the designer, one suggested by certain requirements taken in isolation, etc.). The value of the interdependency assessment activity is precisely that of helping the designer in being broader in his implementation considerations and, therefore, to avoid making early implementation commitments that can (and often do) determine the design.

As it will be pointed out in the next section, one requirement may suggest one implementation technique and a different requirement may suggest another. The two need not even be compatible or appropriate but, as plausible candidates, they should both be taken into account. If this is not done explicitly as advocated here, one of them is likely to be neglected from further consideration and never reconsidered again; in the mind of the designer, since considering the entire design problem is a difficult task, one particular scheme may seem completely inappropriate at a given point in time (maybe, for instance, due to the fact that he is thinking of a specific,

ad-hoc, subset of requirements at that point in time, for which the scheme is clearly a bad alternative). That same scheme may be very convenient for other requirements, but the designer is not likely to reconsider it (in his mind, he already took it into consideration and rejected it; in a complex situation it becomes more and more difficult to realize that something was rejected in different circumstances).

In a sense, the explicit assessment activity attempts to force the designer to being more equanimous: consider all the schemes that occur to you; keep track of them (in form of interdependencies); do not reject any at the outset, leave this to a systematic procedure (the decomposition step) that will be more "neutral" and will suggest to you which ones make real sense for the design problem at hand.

In summary, the assessment activity plays the role of a conceptual discipline whose purpose is to help the designer to cope with the complexity of the design problem. Ideally, this activity should become more structured, and normative guidelines for its execution would help the designer even more. A possibility for reaching nearer that ideal will be briefly discussed in the final section.

3.3. The Interdependency Assessment Activity in Practice: Some Guidelines and Approaches

In this section we focus on the interdependency assessment activity as it appears in practice; we will discuss a couple of apparent problems that may seem to make this activity cumbersome, along with some guidelines that we found useful while carrying it out for the set of requirements of Appendix B; thus, this section is also an account of our own experience with this step of the methodology.

3.3.1. General characteristics of the assessment activity

There are two main issues which one can think of at the outset and which seem to make this activity hard to carry out. They are:

(i) Since we wish to assess interdependencies between all pairs of requirements, isn't this an enormous number of assessments to make (in a set of n requirements, n(n-1)/2 assessments are needed)?

(ii) How does one generate "conceptual models" as suggested above?; how can we be sure that we have considered enough of them?

The first question is legitimate and a matter of concern. With a requirements' set such as that of Appendix B, 3916 assessments must be made (89*88/2). This would mean roughly one hour at one assessment per second, or 60 hours at one assessment per minute. The latter estimate seems more realistic and, consequently, poses a serious question to the practical viability of the process, more so when we realize that such estimates increase as a function of n^2, where n is the number of requirements. It turns out, however, that it is very common to come across pairs of requirements which are obviously unrelated and that such assessments can be made very quickly. As an example, the graph which we obtained for the set in Appendix B

had roughly an average of only 5.6 interdependencies per requirement, which amounts to more than 93% of the assessments being of that "easy" type. The time employed in the assessment activity for that set was roughly 5 hours, or less than 10% of the estimate which assumed an average of one assessment per minute. Although these figures correspond to one specific case, they are similar to others obtained with smaller sets of requirements which we have studied. (See, for instance, the example in [Andreu & Madnick 77.]) Trying to generalize from a limited number of experiences in an empirical way wouldn't be sound, but we believe that one can expect similar circumstances to present themselves in general given the characteristics of complex systems' requirements: The initial set of requirements scanning all or at least most of the aspects of the target system, it is reasonable to expect rather sparse graphs due to the broad spectrum of issues which such a set must cover; from purely use and convenience issues to strictly technical ones.

On these grounds, we would expect the assessment activity to be generally concerned with a large number of "no interdependency" assessments, which will simplify it and speed it up in practice.

The second question, namely, the difficulty of generating conceptual models which provide a basis for detecting interdependencies, is only apparent. The point can be made that any designer makes use of such models, implicitly or explicitly, at some point in the design process; otherwise, one could not end up with a design at all. Many of these models come up naturally while scanning the list of requirements, some are suggested by the requirements themselves, others are the result of the designer's previous experience. Examples of conceptual models useful for the set of requirements in

Appendix B are presented below; they illustrate the circumstances claimed above. The real contribution of the design methodology advocated here regarding these conceptual models is not as much the fact that they must be considered as the idea of doing so explicitly, while keeping track of all of them for the subsequent methodological steps.

3.3.2. Examples of assessments: Some guidelines

In this section we report a number of guidelines which our experience pointed out as useful in actually carrying out the assessment activity. For exposition purposes, it is convenient to organize such guidelines in the following categories: (a) procedural, and (b) generation of conceptual models.

It must be said, at this point, that the interdependencies we assessed for the purposes of this exercise were binary; i.e., any two requirements were seen as either related or unrelated (some of the guidelines below will give hints for a more precise assessment process in which different levels of interdependency would result). The structure given to the requirements set of Appendix B, thus, was an undirected graph without link weights.

(a) Procedural guidelines

These are concerned with how to organize the assessment activity. We found the following useful:

- Establish an order for the assessments to be made. This helps to keep track of the work done and what remains to be done. An easy organization for this purpose is to begin by the first requirement and assess its interdependencies with all the following ones, then take the second, and so forth. This particular organization is useful in another sense: Since one of the requirements in the successive pairs is kept fixed, it is easier to keep in mind the issues, or

"conceptual models" specially relevant to that requirement.

· As successive requirements are considered for possible interdependencies with the fixed one, the person (or persons, see below) making the assessments can think of conceptual models which are relevant to those requirements. It is useful to write these models down along with the relevant requirements for future use. Knowing that they won't be forgotten helps to improve the concentration of the designer(s) on the issues specially relevant to the "fixed" requirement in the current sequence of assessments.

· The same process described in the preceeding point can suggest conceptual models relevant to pairs of requirements for which an assessment has already been made. We found it convenient to treat these as we treated the ones relevant to future assessments, and to consider them later (see below), the reason being that going back to reconsider a previous assessment often disrupts the activity so that some time is needed to center the process again where it was left off.

· Sometimes the conceptual models suggested by the scanning of successive requirements as described above, becomes so numerous that the designer loses concentration. In these cases, we found it useful to change the preestablished order by skipping a sequence of "scheduled" assessments and beginning a new one where the suggested conceptual models are relevant. Later on, the skipped sequence must be considered.

· Sometimes it is useful to skip a sequence of assessments for exactly the opposite reason: it seems that no conceptual models can be generated at all and the designer begins to feel uncomfortable with the assessments, the activity becomes somewhat "dull." We found that in these cases switching to another set of assessments in which other issues are involved helps to match the designer's frame of mind at that point in time to the assessments to be made; the result is a more comfortable feeling regarding the established interdependencies. This, incidentally, points out that this kind of systematic assessment (the designer is conscious that a series of assessments has been skipped and must be reconsidered later on) forces the explicit consideration of all possible interdependencies; in other more flexible circumstances the designer is likely to forget about the ones with which he felt uncomfortable at a given point in time.

· Sometimes the person making the assessments may lose the confidence in himself. The assessments seem to become trivial and too superficial. When these circumstances arise it is advisable to stop the assessment activity for a while, forget about it and return later. Another alternative could be to make ass ments simultaneously by a (probably small) group of persons. Although we have not explored it, this is likely to improve the self-confidence of the persons involved.

· It was suggested above that it appeared appropriate to just take notice of new conceptual models, relevant to assessments already made, and to consider them later. This implies a second

pass over the set of requirements for interdependency assessment purposes. We believe that such a second pass is very useful, since it not only allows the designer to include the new conceptual models that seem appropriate, but it also gives the opportunity of quickly going over the assessments made, thus getting an overall feeling for their correctness and increasing his confidence on them. An interesting observation can be made at this point: it is possible to think of more than one conceptual model in which context two given requirements may be seen to be interdependent. The number of conceptual models relevant in this sense to a pair of requirements could be taken as a measure of the degree of interdependency between them. A more precise assessment scheme could be obtained in this way. Furthermore, the possibility of assigning different "importance" to different models would increase the domain of values that such an "interdependency measure" could take. This alternative opens an area for further research which we believe should be investigated at some depth in a move towards less crude interdependency assessments.

(b) Guidelines for the generation of conceptual models

Although different persons have different conceptual frameworks regarding the organization of knowledge and thus conceptual models in which a pair of given requirements may be seen to present trade-offs or concurrencies will emerge as the result of different mental processes in different individuals, we present here a few instances of the procedures that one can employ to "generate" conceptual models linking requirements in the sense discussed before.

They are taken from the assessment activity that we carried out for the set of requirements in Appendix B. Thus, they illustrate our own experience, and to some extent reflect a specific mental framework; nevertheless, they will serve the purpose of giving an idea of what is involved in actually identifying those conceptual models. From a personal experiential viewpoint, we must say that the models emerged rather naturally from confronting pairs of requirements, and more easily than expected. Although an attempt is made below to categorize the procudures that served to identify these models, the purpose of the discussion is not as much to lay out a set of general normative guidelines as to point out that such guidelines can in fact be developed by each person as experience is gained in making interdependency assessments.

· The first scanning of the requirements' set so as to obtain requirements meeting the conditions discussed in section 2 can already suggest conceptual models useful in the assessment activity. More concretely, removing any implementation connotation from the initial set of requirements is an activity very likely to suggest such models. One should thus look ahead while performing the first scanning of requirements and keep track of potentially useful models. For example, the original (Appendix A) requirement #35

> "Literals being compared to field variables are validated against acceptable field values prior to data search,"

which was removed (see section 2.2.(a)) because it in fact specified a particular implementation, suggests a conceptual model in which requirements #6 and #74 (Appendix B) can be seen to be concurrency

interdependent. Requirement #6 specified:

> "Field definition permits validation of input datum as to acceptable values,"

while #74 read:

> "With a single terminal active, user can receive a response from a direct access to any item in less than 5 secs."

By using the strategy specified by the original requirement #35, access time can be reduced significantly for items whose values are not acceptable. One thus concludes that these two requirements (#6 and #74) are concurrency interdependent, (CI), in the sense that meeting the former can help to meet the latter (note, we say it can, not will: at this point, no commitment is yet made to materialize that concurrency; this only means that it will show up later as an alternative solution to some design subproblem; whether or not it should be adopted in the final design will be a function of other requirements which may end up in the definition of that same subproblem).

· Other CI requirements were identified by looking for a conceptual model in which a possible implementation would allow common processing in the eventual system in order to meet the two requirements involved. As an example, requirements 17 and 19,

> "Data base update can be performed by on-line user through query language," and
>
> "A bulk data base update (initialization) can be performed through system utility,"

call in fact for two "modes" of the same (or very similar) processing.

It is conceivable that common processing might be employed to meet these requirements. Consequently, we assessed them to be CI requirements.

· Another source of CI requirements comes about when two requirements call for somewhat similar functions which must be performed in different circumstances in the eventual system. As an example consider requirements 17 (see above) and 18:

"Data base maintenance can be performed by batch."

There, one may consider that since in an on-line environment the usual priority is quick response time, a possible implementation to meet requirement 17 can only "patch up" the performed update while consolidating updates can be done at the same time as bulk data base maintenance, assuming that such maintenance is performed often enough. In the particular set of requirements that we are analyzing there is no mention of bulk maintenance frequency, but even if there were the above reasoning would be valid: the constraints posed by such a possible requirement specification would come about in other interdependencies, and the decomposition step would take care of their coordination in the context of the overall design.

· Trade-off interdependent (TI) requirements were identifed in a variety of ways. The concept of possible deadlocks in processing concurrent queries, for instance (a concept that any DBMS designer is familiar with), suggests a trade-off between requirement 2,

"Separate files can be defined to be interrelated,"

and 17,

> "Capability to support two or more concurrent queries in different stages of pcoessing."

The former poses constraints to meet the latter in the sense that a "look ahead" operation may be needed in order to avoid a "deadly embrace". This is because two concurrent queries which, although may not seem to need the same files initially, may in fact do so due to established interfile relationships. This becomes more relevant when specific security requirements are in effect (see, for instance, requirement 24). The same concept suggests other TI requirements (like 19 and 23, for example).

· Other TI requirements were identified when a given one posed additional burden on others. An example is the pair of requirements 15,

> "Records in the data base can be changed,"

and 54,

> "User can cancel active request without loss of data integrity."

Obviously, the latter requirement implies more complicated processing for the former, since the changed "record" must be reconstructed if a change query is cancelled; some scheme (the specifics are unimportant at this time) to keep track of what has been done so far in an ongoing query is needed to be able to be backed up effectively if cancelled. Those two requirements were therefore seen as a pair of TI requirements.

· Another common model behind some TI requirements was a need for what can be called "symmetric" processing to meet a given requirement even when it was not explicitly called for by it, but rather by another one. A case in point is that of requirement 26,

"Transaction history facility,"

and 51,

"User can cancel active request without loss of data integrity."

Obviously, in order to keep an accurate transaction history cancelled requests should be accordingly deleted. There are several ways of implementing this, including the possibility of recording a "delete request" transaction, but the point is that the latter requirement poses a "delete" capability requirement in the transaction history which is not implied by requirement 26 alone.

4.- The Resulting Graph. Preliminary Analysis.

The interdependency assessment activity outlined in the preceding section is summarized in Appendix C,(*) which includes a brief description of the motivation (conceptual model) behind each assessment. It generated a 89-node graph which was entered to the decomposition package presented in [Andreu 77(b)]. Appendix D shows how the graph was entered and displays the successive analysis steps performed upon that graph. In the discussion below, references are made to Appendix D while presenting the results of the analysis.

A preliminary analysis was conducted with a two-fold goal: (i) to make sure that all the assessed interdependencies were in fact present in the graph as stored by the decomposition package, and (ii) to detect any possible isolated nodes which the assessment process might have left without links to any other node.

The purpose of the first check is strictly to validate the data entered to the package; at this point it was felt that the package should allow the user to enter, along with each graph link, a brief description of the motivation, or conceptual model, behind it. This is a feature that the current implementation of the package doesn't have and which would be very useful to keep accurate track of the assessment activity, as well as to help in the interpretation of relationships between subgraphs in the final partition; this will become apparent in section 7 below. Thus, this is one of the improvements which can be made to the package in the future.

After making sure that all the assessed interdependencies were accurately stored by the package (the command "NOLK" was used for this purpose, see Appendix D, pages D2 to D4), we employed the command "ISOL" to check for possible isolated nodes (see Appendix D, page D4). It must be pointed out that this type of checking

(*) To facilitate further references, the requirement numbers employed in Appendix C correspond to the parenthesized numbers assigned in Appendix B; see below.

could be done along with the assessment activity, by just keeping track of all the nodes (requirements) for which at least one interdependency has been established. This, however, gets tedious after a while and tends to be forgotten, so that the command "ISOL" was found to be really useful in graphs of non-trivial size.

The check for isolated nodes pointed out that two nodes had been left unconnected. They were the nodes corresponding to requirements 72 and 87 in the set listed in Appendix B:

"System can operate in ordinary office environment,"

and

"Being used at least by 10 users in M_2 months."

It is useful to investigate why these two turned out to be unrelated requirements.

The first one, in fact, should have been deleted at the outset on the grounds that it can be thought of as implied by others. The fact that the DBMS is to be on-line and capable of supporting CRT terminals makes it suitable for office environment, which may require, for instance, a low noise level. Had more specific requirements (for example regarding concrete noise levels and the need for hard copy terminals in the office environment) been present explicitly, requirement 72 would not have ended up being isolated. As the initial set of requirements was set up, however, that requirement doesn't make too much sense and should be neglected.

The second one (#87) is the result of a more fundamental

inconsistency in the initial requirements' set which could not be easily detected before. Since a specific requirement (#86) specifically calls already for a time limit in the system development process, requirement 87 was probably set forth to cover such issues as user training and hardware delivery. Since no other requirement in the set hinges on such issues explicitly, the presence of #87 creates an inconsistency or at least some ambiguity that should be removed. In a real life setting, a designer confronted with this situation should probably go to the users and require more specific information about the meaning of that requirement. In our setting, we decided to remove that requirement from the set we were going to work with.

Consequently, the final requirements' set was further reduced to 87 by the removal of numbers 72 and 87 from the set in Appendix B (see page D5).The remaining requirements were renumbered and assigned consecutive numbers from 1 to 87. The final number assignment is indicated in parentheses in Appendix B.

The interdependencies among these 87 requirements are summarized in Appendix D (pages D6-D8), where the output produced by the command "NOLK" with the reduced set is shown; the average number of links per node was roughly 5.6.

5. A First Graph Decomposition. Concept of Main Subproblems

The graph whose structure is summarized in Appendix D was decomposed with the aid of the package described in [Andreu 77(b)]. The best obtained partition is depicted in Appendix D (pages D10-D11). As it can be seen, only 4 subgraphs were obtained. One of them was rather large (48 nodes), another was sort of "medium size" (21 nodes) and the other two of more reasonable size (8 and 10 nodes respectively).

An analysis of the requirements that ended up in each subgraph pointed out that they referred to fundamental design subproblems that we called "main subproblems" (MS's). For example, subgraph #4 (21 nodes) is concerned with the data manipulation language, and #2 (8 nodes) is centered around the data description language and data items characteristics. It was felt, however, that the obtained subproblems were still too broad in scope to be really meaningful. The idea of decomposing each MS further was considered at this point. We actually carried out such a further decomposition and the results are the subject of the following sections. The remainder of this section discusses the justification for such a further decomposition.

A starting point for the analysis of the obtained 4-way partition is to consider the associated measure value and its components. It is shown in Appendix D (page D11) that the depicted partition has a

strength of 1.069 and a coupling of 0.263, yielding an overall measure of 0.806. The relatively small coupling value is the result of an average of about 10 links between subgraphs out of over 240 links in the graph. This suggested that the obtained subgraphs were relatively decoupled but lacking internal coherence, which, translated into design terms, resulted in very broad MS's. Thus, decomposing each of them further seemed appropriate. We run into a dilemma, though, if we are to take this route: since decomposing each MS further deteriorates (decreases) the overall partition measure, what is the meaning of the further decomposition and why is it meaningful for our purposes? The following paragraphs explore these issues.

The idea of taking each subgraph (MS) in isolation and attempting to decompose them further is in fact equivalent to focusing on the "strength" of each of them. In other words, it means to forget about their interactions and concentrate in their inner structure. From a design standpoint, it implies that the designer (i) considers the broad MS's to be rather independent of one another, and (ii) to gain further insight into their intrinsic structure, he or she decides to analyze each of them in isolation.

Once such a decision is made, what should be the guidelines employed to decompose each MS? To achieve overall consistency in the analysis, the designer should, it seems, attack the decomposition of each MS in the same way as the decomposition of the entire graph was approached to begin with.

This brings up the question of whether evaluating MSs' decompositions in this way will produce any further decomposition or not. It may seem at the outset that since the MS's were identified as the components of the best identifiable partition of the overall graph, each MS would not decompose further; i.e., that the "no partition" solution should be the best solution for each isolated MS.

In general, the above conjecture is not true. The basic reason is that the decision to analyze each MS by itself modifies the decomposition problem. In particular, neglecting the interactions among MS's affects directly the coupling components of the overall decomposition measure. In fact, it can be shown in general (see Appendix E) that the coupling between two subgraphs <u>increases</u> when they are each further decomposed, unless these subgraphs are disjoint to begin with (in which case the coupling is zero either way). By decomposing each subgraph independently, these coupling components are discarded. This shifts the emphasis to the strength components within each subgraph, thus increasing the possibility of finding further decompositions with higher evaluation measures.

Let us explain in more detail why considering each MS in isolation can produce further decompositions with associated measures higher than those found in the original decomposition. Assume that the initial graph is in fact composed of a series of disjoint subgraphs. Each of these subgraphs can be taken in isolation for decomposition purposes without modifying the analysis at all (i.e., the best overall decomposition will be the same as the collection

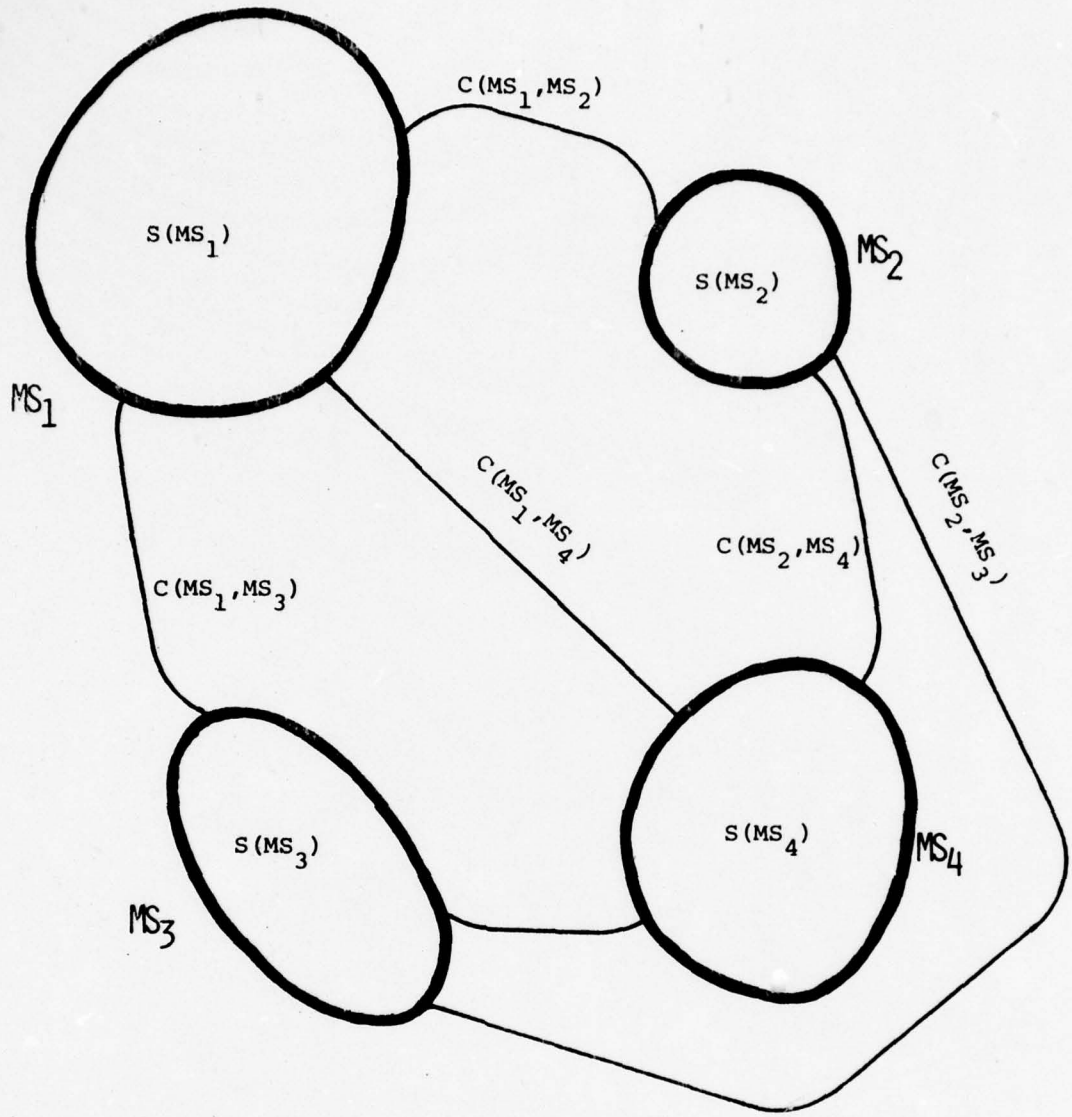
of the best decompositions for each subgraph). However, this does not mean that the best decomposition will have the original disjoint subgraphs as the only components. If such subgraphs are not very compact, they can in turn be decomposed (the opposite would mean that we would never decompose a graph).

From this viewpoint, it is obvious that taking subgraphs (MS's) in isolation for further decomposition is equivalent to setting the coupling measures among them to zero. This can be reasonable if such couplings are relatively low and a more detailed analysis of each MS is wanted. Once this is done, however, the problem is changed.

The foregoing discussion is illustrated schematically in Figures 1 and 2. Figure 1 depicts the result of the original 4-way decomposition of the overall graph as shown in Appendix D (page D11). $S(MS_i)$ is used to indicate the strength of subgraph (MS) i while $C(MS_i, MS_j)$ indicates the coupling between MS's i and j.

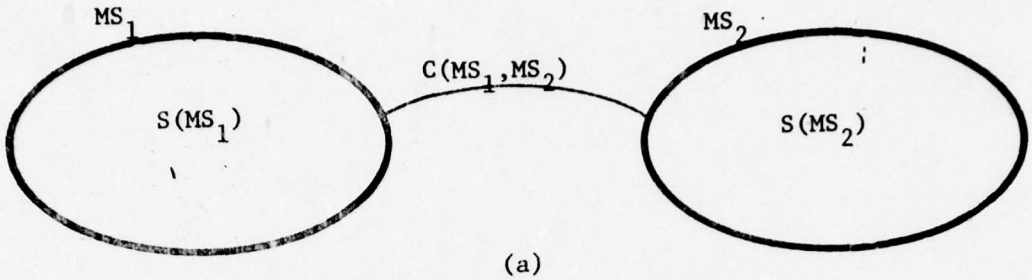
Without loss of generality, let us consider Figure 2, which is a simplified picture with only two subgraphs. Figure 2(a) is the counterpart of Figure 1, that is, the result of the first decomposition. Figure 2(b) depicts what happens to the overall measure if each MS is decomposed further. The coupling components between MS's not only increase in number, but in such a way that their sum is actually higher than the sum of the couplings in Figure 2(a) (see Appendix E). Figure 2(c), finally, depicts how neglecting those coupling terms can result in the identification of further MSs' decompositions.

The following observation should be made at this point. There

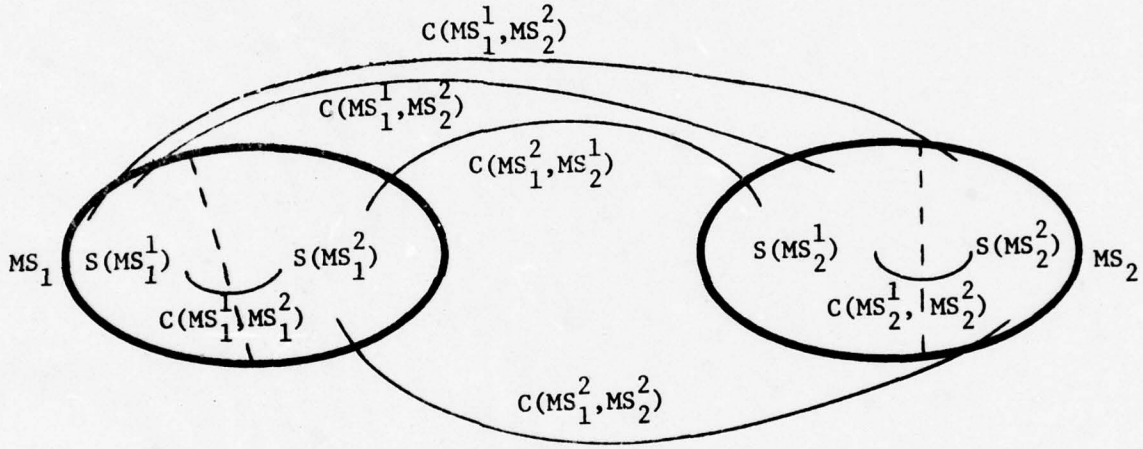


$$M = \sum_{i=1}^{4} S(MS_i) - \sum_{\substack{i=1 \\ j=i+1}}^{4} C(MS_i, MS_j) = 0.806$$

Figure 1


MS1
MS2
C(MS1,MS2)
S(MS1)
S(MS2)

(a)


C(MS1 1,MS2 2)
C(MS1 1,MS2 2)
C(MS1 2,MS2 1)
MS1
S(MS1 1)
S(MS1 2)
C(MS1 1,MS1 2)
S(MS2 1)
S(MS2 2)
C(MS2 1,MS2 2)
MS2
C(MS1 2,MS2 2)

(b)


MS1
S(MS1 1)
S(MS1 2)
C(MS1 1,MS1 2)
S(MS2 1)
S(MS2 2)
C(MS2 1,MS2 2)
MS2

(c)

Figure 2.

are two differences between Figures 2(a) and 2(b), regarding the evaluation of the overall partition. In Figure 2(a), this evaluation would be

$$E_a = S(MS_1) + S(MS_2) - C(MS_1, MS_2) \quad ,$$

while in Figure 2(b) it would be

$$E_b = S(MS_1^1) + S(MS_1^2) - C(MS_1^1, MS_1^2)$$

$$+ S(MS_2^1) + S(MS_2^2) - C(MS_2^1, MS_2^2)$$

$$- \sum_{i,j=1}^{2} C(MS_1^i, MS_2^j).$$

Although (see Appendix E) it is true that

$$\sum_{i,j=1}^{2} C(MS_1^i, MS_2^j) > C(MS_1, MS_2) \quad ,$$

it is also true that

$$S(MS_i^1) + S(MS_i^2) - C(MS_i^1, MS_i^2) > S(MS_i), \quad i=1,2 \quad ,$$

for otherwise the MS's taken in isolation wouldn't have been further decomposed. Therefore, the following question arises: Is it possible that $E_b > E_a$? In other words, is it possible that the evaluation of the overall partition <u>after</u> the second decomposition step be higher than that obtained through the first step?

The answer to this question, in general, is yes; it is possible. However, given the way in which the first partition is generated, it

is highly unlikely. It cannot be said that it is impossible because we did not use an exact optimization procedure to generate the first partition; to the extent that the employed methods are good, however, it is unlikely that the above circumstance will occur. For example, when a second decomposition step is performed upon the 4-way partition shown in Appendix D (page D11), as described in the next section, the overall measure decreases from 0.806 to 0.636.

To recapitulate, there are two basic ideas concerning the decision of taking a second decomposition step:

(1) The MS's obtained in the first decomposition are (i) broad and (ii) relatively loosely coupled.

(2) To achieve better understanding of these broad MS's, the designer can decide to analyze them individually. This is however equivalent to neglecting the couplings among them. Therefore, the results of a second decomposition step in terms of MSs' subparts will present rather involved interactions between subparts of <u>different</u> MS's.

The basic objective of the second decomposition step is thus to gain insight into the inner structure of the MS's; the main motivation for carrying it out is their broad scope as perceived by the designer. It must be kept in mind, however, that since the results are going to complicate the interpretation of interactions among eventual MSs' components, it should only be done if the coupling between MS's is relatively low.

In the next section, such a second decomposition step is

taken. As subsequent sections will illustrate, this permits a clearer understanding of each MS through the collections of design subproblems in which they are decomposed. However, the interactions among these subproblems become more cumbersome. The resulting design framework will thud be in terms of MS's and MSs' components. Its interpretation will point out certain deficiencies in the original requirements set that may be responsible for the MSs being so broad.

6. Analysis of Main Subproblems. The Second Decomposition Step.

As proposed in the preceding section, the four subsets of requirements shown in page D11 were individually treated for further decomposition. Doing so involves creating a decomposition problem for each of those subsets by (1) defining the graph associated with each one as that formed by the nodes in each subset and the set of original links joining nodes in the subset, and (2) computing the associated distance matrix in the usual way. Step (1) was accomplished through the command "DENO" (see page D11) available in the decomposition package by deleting the nodes which don't belong to a particular subset. By making the resulting graph the current graph, the package was then used to decompose it. Since the command "DENO" renumbers the remaining nodes, the interpretation of the obtained decompositions was somewhat inconvenient because it involved tracing back the new node (requirement) numbers to the original ones (those in Appendix B). This suggested a possible improvement for the decomposition package that will be further discussed in section 8. In the remaining of this section we center our discussion on the description of the results obtained from the second decomposition step.

Of the four MS's listed in page D11 , three were further decomposed (numbers 1, 2 and 4). MS 3 was not decomposed because no partition could be found whose measures were superior to those associated with the entire MS.

The results are shown in Appendix D (pages D11 through D14).

For convenience, the final overall partition (Appendix D, page D11) is shown as a function of the original requirement numbers (Appendix B). That decomposition is used as the final design framework, as discussed in the following section.

The final decomposition consists of 13 subsets, whose dimensions vary between 2 and 16. The dimension mean is 6.69, and its standard deviation, 4.16.

To deduce a design framework from these 13 subsets, we analyzed each one of them in terms of their elements (requirements), and examined what interdependencies (links) existed among elements of different subsets. Although we pointed out that interdependencies among MS's could be evaluated through the results of the first partition, it is useful to pay some attention to which MS component (after the second decomposition) most interdependencies among MS's refer to, in order to gain insight as to how different design subproblems interrelate. In the context of our decomposition package, the command "PRLK" applied to the final decomposition is specially well-suited to perform the latter examination (pages D15-D18). To aid in the identification of the main "focus" of each subproblem, it is useful to examine, in each subset, what node (requirement) is involved in more interdependencies. These two simple operations (the second of which is currently only indirectly supported by the decomposition package through "NOLK") allowed us to identify a meaningful design framework which, as discussed in the following section, also pointed out some characteristics of the employed set of requirements (mainly regarding its completeness) that were not detected in earlier analyses.

7. A Design Framework

The design framework (in the form of a collection of design subproblems) presented here was the result of interpreting each subset of requirements in the final partition (Appendix E) as a design subproblem while interpreting the interdependencies among elements of different subsets as coordination requisites for putting those subproblems together.

It was relatively easy to perform these tasks. Each subset turned out to hinge on a rather well-defined design problem which suggested itself without difficulty. The relationships among subproblems also made good sense. However, as we try to emphasize below, when subproblems and relationships are put together in the framework, several interesting issues, not at all obvious in the initial requirements set, began to show up. Such unforeseen results point out the usefulness of the proposed methodology.

Figure 3 depicts the structure of the framework. Boxes indicate the four MS's obtained in the first decomposition step. Inside each box, the corresponding MS decomposition obtained in the second decomposition step is shown; circles in each box represent the design subproblems in which each MS is decomposed. Lines between circles indicate relationships among subproblems, and are labelled according to the main coordination issues. Lines between boxes, finally, represent coordination among MS's; an attempt was made to show which MS components are more heavily involved in each coordination among MS's (dotted lines inside the boxes).

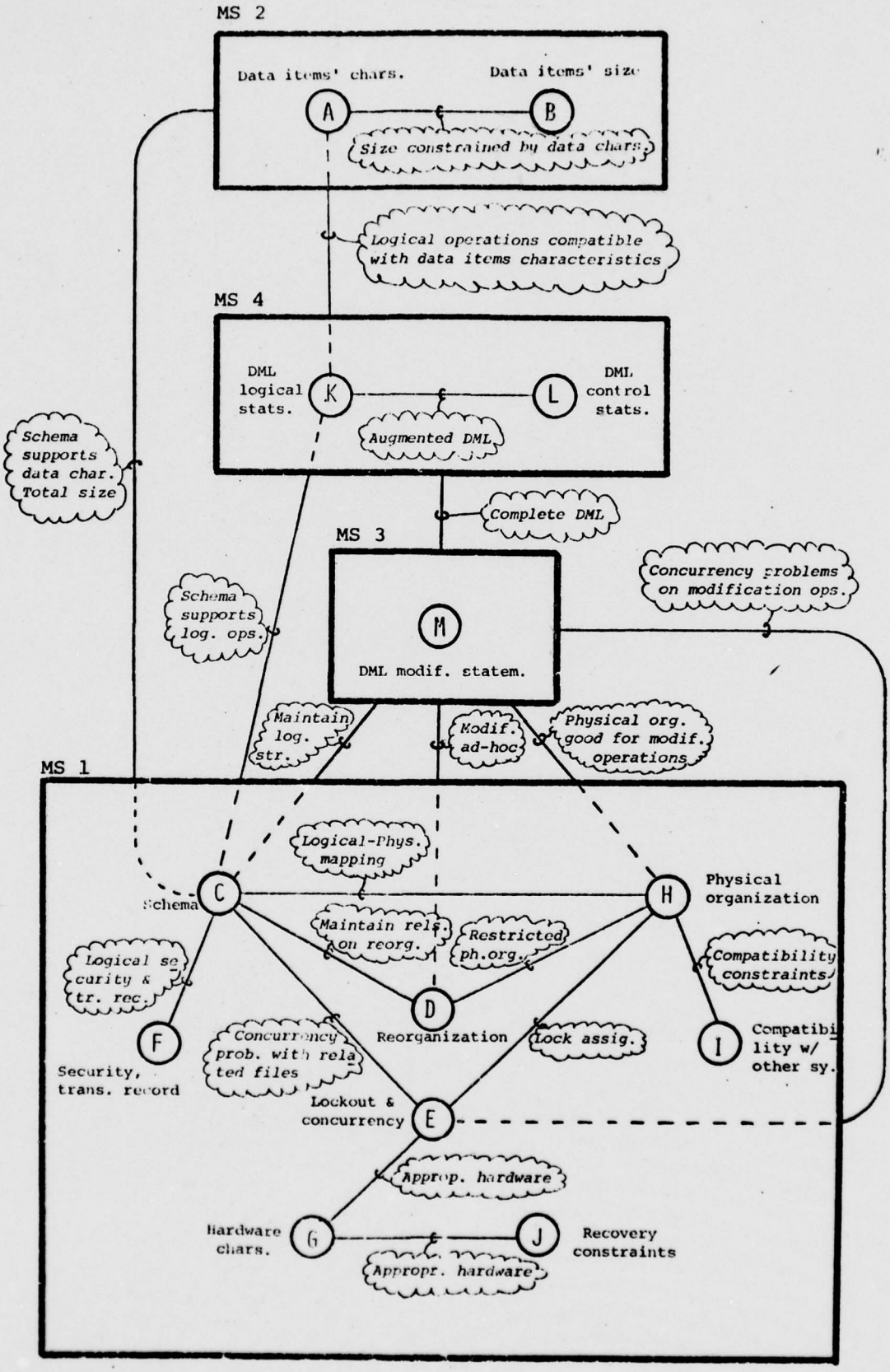

MS 2
Data items' chars.
Data items' size
A
B
Size constrained by data chars.
Logical operations compatible with data items characteristics
MS 4
DML logical stats.
K
L
DML control stats.
Augmented DML
Schema supports data char. Total size
Complete DML
MS 3
Concurrency problems on modification ops.
Schema supports log. ops.
M
DML modif. statem.
Maintain log. str.
Modif. ad-hoc
Physical org. good for modif. operations
MS 1
Logical-Phys. mapping
Schema
C
H
Physical organization
Maintain rels. on reorg.
Restricted ph.org.
Logical security & tr. rec.
Compatibility constraints
D
Reorganization
Lock assig.
F
Concurrency prob. with related files
I
Compatibility w/ other sy.
Security, trans. record
Lockout & concurrency
E
Approp. hardware
Hardware chars.
G
J
Recovery constraints
Appropr. hardware

Figure 3.

In what follows, Figure 3 is analyzed and implications for design are discussed. We begin by a discussion of the framework's general structure.

7.1. General Framework Structure. Main Components.

Three main components can be distinguished in the framework depicted in Figure 3. From top to bottom, they relate to:

a) The characteristics of data items (box 2);

b) Data manipulation language (boxes 3 and 4);

c) Logical data model and physical representation (box 1).

7.1.1 Characteristics of data items

The requirements which ended up in the subset corresponding to MS 2 (Appendix D) all refer to properties characterizing the data items that the system must support. These are of two main types, as it will be further discussed below: one emphasized their "logical" properties (e.g., "null" values can exist); and the other is more physically oriented (e.g., size of data items). The requirement that gives identity to this MS is that calling for a data description language (DDL): it must be such that the data items characteristics can be appropriately described by DDL sentences. This MS was decomposed in two subproblems in the second decomposition step described above; they point out the two types of data items properties just mentioned and will be further discussed below.

7.1.2. Data manipulation language (DML)

MS's 3 and 4 (Appendix D) refer to the need for a DML. It

is interesting to note that the series of requirements calling for the different characteristics of the DML did not end up all in the same MS. There is a good reason for this, when such characteristics are considered in the context of the overall design problem, as follows: certain DML characteristics refer to how data items can be referenced (e.g., by means of relationships among them) while others specify what can be done with the referenced data items (e.g., change or delete). It is apparent that although these characteristics must all be taken into account in the design of the DML, they interact with other requirements (MS's) to very different degrees, so that it makes perfect sense to consider them separately for design structuring purposes. This will become clearer below when the relationships among MS's are discussed.

7.1.3. Logical data model and physical representation.

MS 1 (Appendix D) is the largest of the four and refers to a number of distinct but closely related issues, ranging from the need for a logical data model, to the representation of the data base in physical storage, to the selection of proper hardware. All these issues were assigned to separate subproblems in the second decomposition step, as it will be discussed below. The reason why they were not so in terms of MS's, during the first decomposition, can be explained by the fact that the initial set of requirements was incomplete in a sense that--we believe--can only be detected and checked by an analysis similar to the one being performed here. We will discuss this completeness issue later in section 7.3. Section 7.2.4 is devoted to a detailed discussion of the decomposition

of this MS.

7.1.4. Relationships among MS's.

The relationships among MS's and their meaning for subproblems coordination purposes were interpreted by recalling the "conceptual models" which, in the interdependencies assessment process (see section 3), motivated the assessments that translated into graph links among the subsets of requirements corresponding to the MS's. Such a task required going back to the records of the assessment process in order to retrieve the meaning of each of the relevant links and thus was somewhat time-consuming. A proposal for making this interpretation process more convenient for the desinger, involving the addition of a new feature to the main analysis tool (the decomposition package) was already suggested in section 4 and will be treated in section 8 below.

These relationships are depicted in Figure 3 (page 53) as lines between boxes. Although Figure 3 makes an attempt to show which MS components are more heavily reponsible for the relationships among MS's, such a particularized relationship analysis will be delayed until section 7.2.5, after a detailed description of the components of each MS is performed below. Here, we just note the general characteristics of those relationships in order to complete the general picture of the framework.

· Relationships between MS's 2 and 3: MS 3 is concerned with the "modification" capabilities postulated for the eventual DML (see 7.1.2). Its relationship with MS 2 (data items characteristics) emphasizes the fact that certain data items characteristics (e.g.,

variable length) can complicate the processing of "modification" DML statements (e.g., update).

· Relationships between MS's 2 and 4: MS 4 being concerned with the "logical" capabilities of the eventual DML, including logical operations among data items (e.g., boolean conditionals), its relationship with MS 2 focuses on ensuring that logical operations are compatible with the characteristics of the data items they may involve.

· Relationships between MS's 3 and 4: MS's 3 and 4 referring both to DML characteristics, the relationship between them is merely one of DML completeness.

· Relationships between MS's 1 and 2: This focused on the logical data model (schema) supporting the data items characteristics specified in MS 1 (for example, a provision should be made in the schema for "null" data values) as well as the physical representation of these characteristics; this relationship also brings to MS 1 the data items' sizes to produce an overall data base size.

· Relationship between MS's 1 and 4: This makes explicit the compatibility between DML logical statements and schema structure. It also calls for certain features in the eventual hardware needed to support some DML "control" statements as will be discussed in more detail below.

· Relationships between MS 1 and 3: Most of these emphasize the fact that data base "modification" operations (e.g., change, delete) complicate the physical organization design problem and can be the source of faulty system behavior when concurrency is brought

into play.

* * *

The more detailed inner structure of each MS made explicit by the second decomposition step permits a better understanding of the framework in Figure 3. The following sections are devoted to a more detailed description of each MS.

7.2. Structure of the MS's.

7.2.1. MS 2 (data items' characteristics)

The second decomposition step for MS 2 produced two subproblems, labelled A and B in Figure 3.

Subproblem A focuses on what may be called the "logical" characteristics of the data items to be included in the data base and sets forth the need for a data description language (DDL). From a design standpoint, this subproblem represents a "data analysis" step, necessary for properly defining the kind of information which the target DBMS is to be concerned with. Issues such as checking for compatibility of the different data characteristics and making sure that they are expressable in the eventual DDL are at the heart of this design subproblem. The relationships between this component of MS 2 and certain subproblems in MS 1 pointed out that the employed set of requirements was incomplete; we will discuss this issue at some length in sections 7.2.5 and 7.3 below.

Subproblem B is centered around "physical" characteristics of the data items to be supported, referring mainly to their size as seen by the end user(s); i.e., independently of any physical encoding that may take place for the purpose of representing data items in storage media. This subproblem thus focuses on another aspect of the data analysis step mentioned above, namely, on a preliminary study of the space needed to hold each data item, whose logical characteristics were the subject matter of subproblem A.

The interdependencies among requirements defining subproblems

A and B point out that there is a central coordination issue between the two. Certain logical data items characteristics may pose constraints on their size which may help to give a more accurate picture of the size range for the associated item. For example, the existence of a range of acceptable values for a given data item (its domain) gives additional information about its size in a more specific way. Furthermore, the relationships among these subproblems and others that come about in the decomposition of MS 1, make explicit certain encoding techniques which may be used to represent data items in storage, as it will become clear in section 7.2.5 below.

In summary, MS 2 represents the initial data analysis step needed in the design of a DBMS aimed at supporting applications in a specific setting. The distinction made between logical and physical (size) data items characteristics in the sense above is meaningful from a design structuring viewpoint because it points out the possibility of attacking the associated subproblems independently as long as the coordination issues sketched in the preceding paragraphs are not neglected: The design of a DDL capable of describing any possible logical data characteristic is a problem whose main emphasis is very different from that aimed at pinpointing the specific size constraints for each data item. One can treat these two problems one at a time, provided that sufficient attention is paid to their proper coordination; the coordination, as suggested above, goes mainly from subproblem A to subproblem B, i.e., it is the size of a data item that is a consequence of its logical characteristics or properties rather than the other way around. Coordination in the other direction is also conceivable but not as strong; for example, the DDL design must take into account

the size range of each item in order to allow enough space in its constructs so as to be able to express instances of data items without much problem. This would mean that if, for whatever reasons, subproblems A and B can't be attacked simultaneously (but, note, still almost independently of one another) in the design process, subproblem A should probably be considered first.

7.2.2. MS 4 (DML)

MS 4 was decomposed into two subproblems, labelled K and L in Figure 3.

As advanced in 7.1.2 above, MS 4 is concerned with data manipulation language (DML) issues, particularly with those regarding data selection (as opposed to data modification). Subproblems K and L both refer to such issues but have different connotations. Subproblem K may be called a "traditional DML design problem" in the sense that it is centered around typical DML constructs, such as types of references to data items and allowed manipulations (logical operations) of data items or collections of data items (e.g., relational conditionals). In this sense, subproblem K centers on providing a DML capable to supporting the required operations among data items and their references, and, accordingly, is related to subproblem A to ensure that the different operations match the characteristics of the involved items (see 7.2.5).

Subproblem L, on the other hand, is concerned with a class of DML statements that may be called "control" statements. These are capabilities of the DML which are not as "standard" as the ones

considered in subproblem K (e.g., report control, requirement 42). In the context of the overall design problem, viewing these DML requirements as a separate subset helps to understand better their interactions with other design issues such as the selection of hardware devices; this is discussed more thoroughly in section 7.2.5.

The relationship between subproblems K and L is merely one of DML completeness. In the (logical) design of the DML, "control" statements and more traditional manipulation statements can be treated similarly; however, when coordinating that design subproblem with others, the control statements pose specific constraints on certain subproblems' structure which wouldn't appear as clearly if the distinction between the two types of statements were not made. This makes the coordination between subproblems K and L rather trivial and points out an interesting characteristic of the final decomposition: Subsets of requirements are formed (i.e., design subproblems are defined) not only as a consequence of similarity among their elements (i.e., of trade-offs or concurrencies among their component requirements) but also as a consequence of similar relationships with other requirements (defining other design subproblems). This makes good sense from a design standpoint because it makes subproblem coordination clearer.

7.2.3. MS 3 (DML modification statements).

MS 3, which, as noted in section 7.2.2 completes the DML design problem with MS 4, was not decomposed during the second decomposition step performed upon the subsets listed in Appendix D. Its

main emphasis is in DML modification statements; its relationship with MS 4 is, again, DML completeness. As it happened with subproblems K and L in MS 4, the clustering of requirements calling for DML modification statements in a separate subset from the rest of DML requirements is due to their relationships with other subproblems. It is apparent that the implementation of data base modification facilities is heavily dependent upon the kind of physical data base organization eventually employed. For example, the need for supporting updates may rule out several physical organizations which could be excellent choices if, say, only retrival operations were to be supported. It is important to make these interactions explicit in the structure of the overall design problem. The reason for MS 3 is thus not as much the fact that the requirements defining it imply a clearly separated DML design subproblem, but rather that they exhibit specific relationships with other subsets of requirements which are not present in the remaining DML specifications.

An interesting feature is to be noted in the subset of requirements defining MS 3: Requirement 51 (Appendix B) ended up in that subset while it seems to have all the characteristics of a DML "control" statement requirement and thus one would expect to find it in the subset corresponding to subproblem L, MS 4 (see 7.2.2). There is a good reason, though, for assigning requirement 51 to MS 3: it implies a series of potential operational problems which become important only when applied to data base modification operations. For example, cancelling a retrieval request poses no serious problem, but cancelling a, say, update request is not so easy. In this sense, including requirement 51 in MS 3 helps to make explicit

certain implementation issues which wouldn't be as apparent had it been assigned to subproblem L as could be expected.

7.2.4. MS 1 (logical data model, physical representation)

For reasons which will become clear below, MS 1 resulted in being a very broad MS. Consequently, it was the one which most benefitted from the second decomposition step. It was decomposed into 8 subproblems, labelled C through J in Figure 3 (page 53). We discuss them (and their relationships) in turn below.

· Subproblem C focuses on the need for a logical data model and the overall data base size. The data model includes security considerations at the logical level. It corresponds to the design problem concerned with the identification of an appropriate data model to support the structure of the information that the target DBMS is to deal with, including relationships among "files" that the initial requirements called for explicitly.

The schema design subproblem is one that would generally be expected to show up very clearly in a DBMS design problem. In our context, it is surprising that this subproblem didn't pop up as an MS in the first decomposition step. One explanation is the following: At the interaction between MS's level, the relationship between MS's 1 and 2 (see 7.1.4) focuses on the schema supporting the logical characteristics of the data items in the data base. For example, "null" values and domains are to be supported. However, the initial set of requirements wasn't too explicit about data items' charateristics (for instance, no requirement specifies that different

"types" of data items must be supported, let alone what these types are). Had it been so, the obvious interdependencies between data item characteristics and schema structure would have been established during the assessment process and the set of requirements defining the schema design problem could have been distinguished in a way similar to that which produced MS 3 (i.e., by way of similar relationships to other requirements). In this sense, it can be said that the initial set of requirements was incomplete.

· Subproblem F focuses on the need for a transaction recording facility and signon security. The presence of these two issues in the same subproblem suggests that performing security control and recording transactions are to be considered at the same time for design purposes. This makes sense since both operations need to analyze the transaction to be processed and are interdependent (e.g., a transaction can be rejected on security grounds and thus shouldn't be recorded). Furthermore, this subproblem is related only to subproblem C, which suggests that transaction recording and security checking should be performed at the logical level. This also makes good sense since doing it at the physical level, given that physical reorganization is possible, would be rather cumbersome.

· Subproblem H emphasizes the problem of physical organization and has explicit constraints regarding file size and response time. An interesting consideration to be made here is that the physical organization design is heavily centered around data base size. Our experience tells us that it is very common to take query frequency

into account in such a problem. At this point we realize that no reference to query frequency appears in the initial requirements set. This is, we feel, another source of incompleteness in the original set of requirements.

It is interesting to notice also that the two requirements regarding costs (development and maintenance) ended up in this subproblem. Maintenance costs should be considered here in order to avoid physical structures difficult to maintain. The reason for the presence of the development costs requirement is not as clear. In some sense, this should be everywhere. It can be argued that to the extent that subproblem H is related to both hardware selection problems (see below) and to logical structure issues, it is a reasonable place for it to appear.

This subproblem presents well-defined relationships with subproblems C, D and I. Its interaction with C makes the data independence requirement explicit and thus points out the issues regarding logical-physical mapping. (The other interactions are discussed below).

· Subproblem I is concerned with system compatibility with other systems. It is related only to H and thus represents compatibility constraints that the eventual physical organization must meet. In isolation, this subproblem must be seen as one focusing on making sure that the different compatibility constraints don't interfere with one another (i.e., that they can be met simultaneously).

· Subproblem D focuses on the requirements calling for data-base reorganization facilities and their motivation. Note

that requirement 80 ended up in this subproblem, suggesting that it can be achieved through reorganization. Its interactions are with subproblems C and H. The former emphasizes the need for maintaining the same schema while reorganization takes place; the latter constraints the physical organization design problem in the sense that the eventual organization must be well suited to support the kind of reorganization required.

· Subproblem E is concerned with lockout and concurrency issues. The fact that these two issues showed up together is meaningful: depending on the lockout level the problems posed by concurrent data base usage vary a great deal. Its interactions are with subproblems C, H and G. The first is a result of the requirement calling for the existence of related "files" in the schema; two queries which in principle may not seem to refer to the same data items may in fact do so through interfile relationships, so that unforeseen concurrency problems may arise. The second emphasizes the issue of lock assignment to physical structures. The third is discussed below.

· Subproblem G is the central hardware problem. In addition to setting up constraints for the type of devices that may eventually be employed, in terms of capacity, etc., it presents relationships with subproblem E emphasizing the need for hardware capable to supporting concurrency reasonably well. It also relates to subproblem J as discussed below.

· Subproblem J is concerned with recovery constraints. Its relationship with subproblem G passes these along to the hardware

selection problem.

7.2.5. Relationships among MS's components.

Now that the structure of each MS has been analyzed in some detail and their components are known, it is useful to re-examine the relationships among MS's from the viewpoint of which of their components are more heavily involved in those relationships. The discussion in this section is thus an expansion of that in 7.1.4 and puts emphasis on the relationships involving MS 1, the one with the most complex structure.

· Relationships between MS's 1 and 3. Subproblem M, the only component of MS 3 focusing on DML modification statements relates to the following subproblems in MS 1: C, D, E and H.

The interaction with subproblem C materialized through the requirement calling for the existence of interfile relationships and thus is centered around the idea of maintaining these relationships upon data base modification; in other words, it makes explicit the fact that whenever the value of a data item involved in an interfile relationship is modified, care should be taken to ensure that the relationship is properly maintained.

The interaction with subproblem D, which focuses on physical data base reorganization, materialized through an inter-dependency suggesting that the batch data base maintenance facility can in fact be used to make definitive any changes that have been made to the data base while in normal operation since the last batch maintenance operation. Thus, it suggests the possibility of

processing data base moficiations in an ad hoc manner during on-line operation in order to improve their processing time, by "patching up" the current physical organizations and leaving their materialization to the off-line maintenance processing. This broadens the range of possible implementation alternatives for the processing of modification queries.

Interaction with subproblem E, concerned with concurrency requirements, points out that data base integrity may be at odds when processing modification queries in a concurrent environment and therefore calls attention to the problems which may result with careless implementation of such queries.

Interaction with subproblem H, finally, poses constraints on the alternatives for physical organization, by pointing out that certain alternatives may not support the required modification operations well enough.

· Relationships between MS's 1 and 4. The analysis of the graph links joining the elements of MS's 1 and 4, and their associated interdependency assessments pointed out that subproblems C and G were the most heavily involved in interactions with MS 4. The former points out that the eventual schema must support the kind of logical operations and data references called for by the required manipulations and retrieval operations. The latter defines an interaction between required DML control statements and hardware characteristics, pointing out the need for peripheral equipment capable of allowing the generation of control signals that the target system can then catch and process as required.

· Relationships between MS's 1 and 2. These were materialized mainly to subproblems C and H, thus emphasizing the need for the schema to support the data items characteristics defined in subproblem A and the concern for their encoding in sotrage representations; the data items expected sizes pose constraints upon the amount of storage needed.

The comparatively simple structure of MS's 2, 3 and 4 makes unnecessary any further discussion of their interactions (see 7.1.4).

7.3. Comments on Some Characteristics of the Employed Requirements Set

As mentioned in the preceding section, the final subproblem structure pointed out certain characteristics of the requirements set that we employed which had not been detected previously. They related mainly to the requirements' completeness, and were, we believe, responsible for a very complex MS in which distinct subproblems were not apparent until the second decomposition step was performed. The way in which such characteristics were detected points out two main properties of the methodology advocated in this paper which we think are worth emphasizing.

a) It wasn't in any formal way that those characteristics were made apparent; rather, it was our previous experience with similar problems that pointed out that certain requirements were in fact missing. This shows that the advocated methodology doesn't attempt to suppress the designer as such; on the contrary, his intuition and expertise play a central role. The methodology is only a way of organizing the designer's work while trying to make it less painful in activities which can be approached in a formal way (e.g., in the decomposition step).

b) As advanced in the methodology proposal (see [Andreu & Madnick 77]), the final subproblem structure pointed out inconsistencies in the requirements set, so that, at this point, the methodological steps, beginning at the establishment

of requirements should be repeated; the application of the methodology becomes in this sense iterative. In our case, the lack of requirements specifically stating the types of data items to be supported, and the lack of information about query frequency motivated that certain design subproblems were atypical and became less apparent than they otherwise could have been. It is interesting to notice that these missing requirements make the set ill-suited even for its original purpose, DBMS selection (for example, the application might need say character string data items; the way in which the requirements' set was established makes it impossible to check for such a capacity in a candidate DBMS).

In summary, the requirements set of Appendix B is far from the ideal. We believe that correcting the detected inconsistencies would result in a more comprehensible design problem structure than that schematically depicted in Figure 3. Nevertheless, the exercise reported here did show the usefulness of the proposed methodology and, furthermore, it did produce a design problem structure which was not obvious from the original requirements set.

8. Comments on the Usefulness of the Decomposition Package and Suggestions for Improvements

In general, the decomposition software package described in [Andreu 77(b)] was found reasonably appropriate to conduct the analysis discussed above. There were a few facilities that we would have liked to have, however, because they would have made the analysis task more convenient. We have already mentioned them throughout this report at the points where their need was more strongly felt during the analsyis, but we repeat them here as a proposal for package improvement which could be undertaken.

· The need for a more specific description of each requirement than that provided by a requirement number arose several times throughout the usage of the package. The possibility of allowing a short requirement description to be stored, which could be retrieved by specifying the associated requirement number, would be very helpful particularly during the subproblem and interaction interpretation activity.

· The fact that the "DENO" command, used to delete requirements from the "current set," renumbers the remaining ones so as to maintain a series of consecutive numbers, makes working with reduced sets (e.g., during the second decomposition step) rather cumbersome. Being capable of using the original requirement numbers in these situations would make life easier.

· Interpreting the relationship or interactions among resulting subproblems requires consulting the motivation for the interdependencies among the subsets of requirements corresponding to those subproblems. This interpretation task would be made more convenient if the package allowed the assignment of a short description to each assessed interdependency, that could be subsequently retrieved.

· For the purpose of identifying the main focus of each final subproblem, finally, we found it useful to look for the requirement in the associated subset which was involved in the largest number of interdependencies. A command that would perform this type of checking for the "current requirements set" could prove useful for those purposes.

These improvements on the decomposition package don't look too difficult to implement in principle (although they would require considerable disk space on the PRIME) and can improve the package conveniency significantly, particularly when requirements sets of non-trivial size must be analyzed.

9. Areas for further research

At this point, we believe it is fair to say that the experiment discussed above has provided us with valuable insights as to the practical viability of the methodology investigated, as well as with a set of guidelines for the execution of the interdependency assessment activity. However, since, as pointed out in section 7.3 above, the results have been less conclusive than they could have been with a more complete set of requirements, we plan to redo the analysis after completing the requirements set in the areas suggested by the exercise. This constitutes the short term plan for our research activities.

From a more long term planning standpoint, there are two main areas that seem worth investigating: (1) upgrading the decomposition package in order to make it more convenient to use as suggested in section 8 above, and (2) a search for a more systematic approach to the interdependency assessment activity. The issues involved in the former are rather straight forward and should be clear from the discussion in section 8. The latter, although it may seem a dream at this stage, given its characteristics of personal designer's involvement, may, we believe, be approached by relying on a formal requirements statement language (e.g., PSL, see [ISDOS 75]). It may be possible to take advantage of the logical constructs built into such languages for the purpose of identifying rules for the definition of interdependencies. This new possibility, however, cannot be pursued within the time and scope of this project.

REFERENCES

[Andreu & Madnick 77]: Andreu, R. and Madnick, S.E.: "A systematic approach to the design of complex systems; Application to DBMS design and evaluation", C.I.S.R. Report No. 32, M.I.T. Sloan School of Management, March 1977.

[Andreu 77(a)]: Andreu, R.: "Set decomposition: Cluster analysis and graph decomposition techniques", Technical Report #1, project N00039-77-C-0255, September 1977

[Andreu 77(b)]: Andreu, R.: "Solving decomposition problems: Alternative techniques and description of supporting tools", Technical Report #2, project N00039-77-C-0255, September 1977.

[ISDOS 75]: "Problem Statement Language (PSL) user's manual", ISDOS project, The University of Michigan, Ann Arbor, March 1975.

APPENDIX A

ORIGINAL SET OF REQUIREMENTS

1. Data base schema (data dictionary) including interfile relationships, is defined and maintained independently of data base usage.

2. Separate files can be defined to be interrelated.

3. Data description language is English-like and self-documenting.

4. Data base schema is validated by system prior to usage.

5. Interfile relationships can be described at run time.

6. Field definition permits validation of input datum as to acceptable values.

7. The maximum number of files in the data base is at least 10.

8. Maximum number of interrelated files is at least 5.

9. Maximum size of a logical record is at least 500 information characters.

10. Maximum size of an item (field) is at least 100 characters.

11. Maximum number of items in a record is at least 50.

12. Repeated fields (multi-valued attributes) can be defined.

13. Variable sized fields can be defined.

14. Records in the data base can be added, changed or deleted.

15. Data base update can be performed by on-line user through query language.

16. Data base maintenance can be performed by batch.

17. A bulk data base update (initialization) can be performed through system utility.

18. Null-value field generation and identification supported.

19. Field update can trigger computation of a correlated field in same record.

20. Field update can trigger computation (e.g., tally of a correlated field in different record/file).

21. Data integrity supported at least at file level lockout on update.

22. Record level lockout.

23. Checkpoint/restore facilities.

24. Transaction history facility.

25. Separate security privileges for retrieval and update.

26. Data base level security.

27. File level security.

28. Field level security.

29. File maintenance facilities are available (such as file reorganization, compression, copy, dump, etc.).

30. Non-procedural user's language available for on-line query and update.

31. Boolean and relational conditionals can be used in record selection criteria.

32. Arithmetic expressions can be used in record selection criteria.

33. Text string scanning expressions can be used in record selection criteria.

34. Relational condition can compare variable to variable.

35. Literals being compared to field variables are validated against acceptable field valies prior to data search.

36. Data meeting selection criteria (or their pointers) can be used for subsequent query processing.

37. Selected data can be sorted by at least one sort key.

39. Capability for report formatting.

39. Report formatting optionally automatic.

40. Report break control feature supported.

41. Report summary line feature supported.

42. Multi-valied fields can be selectively listed.

43. Procedures can be written which interact with data base and on-line terminals.

44. Procedures interactively created.

45. Procedures can be parameterized and pre-stored.

46. Procedures can utilize user-written subroutines.

47. Screen (menu) formatting facilities supported.

49. Local keyboard terminal supported.

50. CRT terminal supported.

51. Delta Data 5260 supported.

52. User can interrogate status of system.

53. User can interrogate status of current request.

54. User can cancel active request without loss of data integrity.

55. User can suppress listing, save report and later reinitiate listing.

56. User can direct output to a system printer.

57. User can route listing to other terminal.

58. Capability to broadcast messages to all terminals.

59. Signon Security.

60. A master terminal facility with privileged commands and control is supported.

61. Master terminal can be relocated to any on-line terminal.

62. System set-up effort and each subsequent SYSGEN less than one man-month.

63. Utilities to aid set-up.

64. Average up-time for the minimum configuration of at least 95% over a 30-day period.

65. Average system recovery time is 2 hours over a 30-day period.

66. Maximum recorvery time is 24 hours.

67. Two hour on call maintenance must be available.

68. Maintenance requirements less than 1hour/week.

69. Power fail restart capability.

70. Dual processor fail soft capability.

71. Removable disks containing data base can be mounted and processed during an on-line session.

72. A job accounting recording facility is supported sufficient to charge users by application and by department.

73. A job accounting reporting facility.

74. Application is transportable to/from the Agency's existing systems.

75. Data is transportable to/from the Agency's existing systems.

76. System can operate in ordinary office environment.

77. System can communicate with other Agency's systems.

78. With a single terminal active, user can receive a response from a direct access to any item in the data base in less than 5 secs.

79. Response time not a function of file size.

80. Capability to support at least 10 active terminals.

81. Capability to support two or more concurrent queries in different stages of processing.

82. Display rate of terminal at least 120 ch/sec on a CRT and 15 ch/sec on a hard copy terminal.

83. Dynamic control of working storage requirements within program region.

84. Dynamic relocation of programs to avoid "checkerboarding" of usable memory.

85. Additional memory will enhance performance.

86. Dynamic file reorganization capability.

87. Dynamic reallocation of released and deleted storage areas.

88. Disks are removable.

89. File sizes are limited only by disk storage capacities.

90. Total on-line data base can be distributed over many disk units.

91. Controls for tuning system performance at SYSGEN or system load time.

92. Controls for dynamically tuning system performance during run time.

93. Application tuning can be accomplished by restructuring the data to bias retrieval vs. update performance characteristics.

94. Can be delivered within M_1 months.

95. Being used by at least 10 users in M_2 months.

96. Non-recurring costs for basic configuration not more than C.

97. Maintenance costs less than MC/month.

APPENDIX B

MODIFIED SET OF REQUIREMENTS (*)

(1) 1. Data base schema (data dictionary) including interfile relationships, is defined and maintained independently of data base usage.

(2) 2. Separate files can be defined to be interrelated.

(3) 3. Data description language is English-like and self-documenting.

(4) 4. Data base schema is validated by system prior to usage.

(5) 5. Interfile relationships can be described at run time.

(6) 6. Field definition permits validation of input datum as to acceptable values.

(7) 7. The maximum number of files in the data base is at least 10.

(8) 8. Maximum number of interrelated files is at least 5.

(9) 9. Maximum size of a logical record is at least 500 information characters.

(10) 10. Maximum size of an item (field) is at least 100 characters.

(11) 11. Maximum number of items in a record is at least 50.

(12) 12. Repeated fields (multi-valued attributes) can be defined.

(13) 13. Variable sized fields can be defined.

(*) Paranthesized numbers correspond to the final assignment after requirements 72 and 87 were deleted (see section 4, page 38).

(14) 14. Records in the data base can be added.

(15) 15. Records in the data base can be changed.

(16) 16. Records in the data base can be deleted.

(17) 17. Data base update can be performed by on-line user through query language.

(18) 18. Data base maintenance can be performed by batch.

(19) 19. A bulk data base update (initialization) can be performed through system utility.

(20) 20. Null-value field generation and identification supported.

(21) 21. Field update can trigger computation of a correlated field in same record.

(22) 22. Field update can trigger computation (e.g., tally of a correlated field in different record/file).

(23) 23. Data integrity supported at least at file level lockout on update.

(24) 24. Record level lockout.

(25) 25. Checkpoint/restore facilities.

(26) 26. Transaction history facility.

(27) 27. Separate security privileges for retrieval and update.

(28) 28. Data base level security.

(29) 29. File level security.

(30) 30. Field level security.

AD-A048 765 ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MASS C--ETC F/G 9/2
AN EXERCISE IN SOFTWARE ARCHITECTURAL DESIGN: FROM REQUIREMENTS--ETC(U)
NOV 77 R C ANDREU, S E MADNICK N00039-77-C-0255
UNCLASSIFIED CISR-P010-01-05 NL

2 OF 2
AD
A048 765

END
DATE
FILMED
7-78
DDC

(31) 31. Non-procedural user's language available for on-line query and update.

(32) 32. Boolean conditionals can be used in record selection criteria.

(33) 33. Relational conditionals can be used in record selection criteria.

(34) 34. Arithmetic expressions can be used in record selection criteria.

(35) 35. Text string scanning expressions can be used in record selection criteria.

(36) 36. Relational condition can compare variable to variable.

(37) 37. Data meeting selection criteria can be used for subsequent query processing.

(38) 38. Selected data can be sorted by at least one sort key.

(39) 39. Capability for report formatting.

(40) 40. Report formatting optionally automatic.

(41) 41. Report break control feature supported.

(42) 42. Report summary line feature supported.

(43) 43. Multi-valued fields can be selectively listed.

(44) 44. Screen (menu) formatting facilities supported.

(45) 45. Local keyboard terminal supported.

(46) 46. Remote keyboard terminal supported.

(47) 47. CRT terminal supported.

(48) 48. Delta Data 5260 supported.

(49) 49. User can interrogate status of system.

(50) 50. User can interrogate status of current request.

(51) 51. User can cancel active request without loss of data integrity.

(52) 52. User can suppress listing, save report and later reinitiate listing.

(53) 53. User can direct output to a system printer.

(54) 54. User can route listing to other terminal.

(55) 55. Capability to broadcast messages to all terminals.

(56) 56. Signon Security.

(57) 57. A master terminal facility with privileged commands and control is supported.

(58) 58. Master terminal can be relocated to any on-line terminal.

(59) 59. System set-up effort and each subsequent SYSGEN less than one man-month.

(60) 60. Utilities to aid set-up.

(61) 61. Average up-time for the minimum configuration of at least 95% over a 30-day period.

(62) 62. Average system recovery time is 2 hours over a 30-day period.

(63) 63. Maximum recovery time is 24 hours.

(64) 64. Maintenance requirements less than 1 hour/week.

(65) 65. Power fail restart capability.

(66) 66. Dual processor fail soft capability.

(67) 67. Removable disks containing data base can be mounted and processed during an on-line session.

(68) 68. A job accounting recording facility is supported sufficient to charge users by application and by department.

(69) 69. A job accounting reporting facility.

(70) 70. Application is transportable to/from the Agency's existing systems.

(71) 71. Data is transportable to/from the Agency's existing systems.

72. System can operate in ordinary office environment.

(72) 73. System can communicate with other Agency's systems.

(73) 74. With a single terminal active, user can receive a response from a direct access to any item in the data base in less than 5 secs.

(74) 75. Response time as independent as possible of file size.

(75) 76. Capability to support at least 10 active terminals.

(76) 77. Capability to support two or more concurrent queries in different stages of processing.

(77) 78. Display rate of terminal at least 120 ch/sec on a CRT and 15 ch/sec on a hard copy terminal.

(78) 79. Dynamic file reorganization capability.

(79) 80. Dynamic reallocation of released and deleted storage areas.

(80) 81. File sizes are limited only by disk-storage capacities.

(81) 82. Total on-line data base can be distributed over many disk units.

(82) 83. Controls for tuning system performance at SYSGEN or system load time.

(83) 84. Controls for dynamically tuning system performance during run time.

(84) 85. Application tuning can be accomplished by restructuring the data to bias retrieval vs. update performance characteristics.

(85) 86. Can be delivered within M_1 months.

87. Being used by at least 10 users in M_2 months.

(86) 88. Non-recurring costs for basic configuration not more than C.

(87) 89. Maintenance costs less than MC/month.

APPENDIX C

INTERDEPENDENCIES AMONG REQUIREMENTS

(NOTE: Below, (T) indicates trade-off; (C) concurrency).

* Requirement 1 is related to:
  - 2 (C): Interfile relationships included in schema.
  - 4 (C): Chance of validating it without interferring with usage.
  - 5 (T): Schema may become inefficient if already bound.
  -18 (C): Common processing conceivable.
  -19 (C): Common processing possible.
  -27 (T): Points out that "independently of usage" means only "while data base is being used", as oposed to type of usage.
  -59 (C): Being independent of usage simplifies things.
  -60 (C) : Common processing conceivable.

* Requirement 2 is related to:
  - 5 (T): Interference with usage.
  - 7 (T): Increases possible number of interfile relationships.
  - 8 (T): Poses constraints.
  -12 (T): Interrelationshps made more complicated.
  -14 (T): Relationships' maintenance made more difficult.
  -15 (T): See 14.
  -16 (T): See 14.
  -22 (C): Relationship information can be used for this purpose.
  -23 (T): Related files complicate maintaining integrity.
  -25 (T): Made difficult by (2).
  -29 (T): Difficult to enforce by (2); new "conditions" generated.
  -33 (C): Basis for supporting it in (2).
  -51 (T): Back up made harder.
  -73 (C): Explicitly maintained relationships may speed up process.
  -76 (T): "Look ahead" may be needed to prevent deadlocks.
  -78 (T): Relationships must be maintained.
  -82 (C): Relationships can be used for tuning purposes.

* Requirement 3 is related to:
  - 6 (T): Acceptable field values must be expressable in DDL.
  -12 (T): Supported by DDL.
  -13 (T): Puts burden on DDL.

-20 (T): Expressable in English-like language.

-21 (T): Difficult to express in DDL.

-22 (T): See 21.

-27 (T): Puts conditions on security declarations.

-31 (T): DML compatible with structure defined by DDL.

* Requirement 4 is related to:

- 5 (C): Validation of new relationships can be made more quickly.

* Requirement 5 is related to:

- 8 (T): Poses constraints on (5).

-14 (C): Common processing possible.

-15 (C): See 14.

-16 (C): See 14.

-17 (C): Defining relationships may be necessary on updating.

-83 (C): Can be used for tuning.

-84 (C): Restructuring by changing relationships possible.

* Requirement 6 is related to:

-20 (C): Facilitates checking of whether null value is allowed or not.

-21 (C): Can facilitate catching integrity violations on result.

-22 (C): See 21.

-23 (C): Facilitated by (6).

-73 (C): As suggested by old requirement 35 (see section 3.3.2).

-74 (C): Similar to (73).

* Requirement 7 is related to:

-10 (T): Together imply constraints for total size.

-19 (C): (7) can be more reasonably met.

-23 (C): More files can permit better "granularity" for locks.

-27 (C): More files may allow definition at file level.

-29 (T): Additional burden on security.

-76 (C): Facilitated; increased probability of two user working with different files.

-81 (C): Facilitated by increasing number of files; possibility of separating files on disk.

* Requirement 9 is related to:
  -10 (T): Constraints on total size.
  -11 (T): See 10.
  -12 (T): See 10.
  -13 (T): See 10.

* Requirement 10 is related to:
  -11 (T): Total size constraints.
  -12 (T): See 11.
  -13 (T): See 11.

* Requirement 11 is related to:
  -12 (T): See above.
  -13 (T): See 12.

* Requirement 12 is related to:
  -13 (C): Similar representation conceivable.

* Requirement 13 is related to:
  -15 (T): Made difficult; may require reorganization.
  -20 (C): Less space necessary for null values a possibility.
  -21 (C): Included in same processing ("look ahead") may speed things up.
  -22 (T): Other field may be also variable in length, which makes processing worse.

* Requirement 14 is related to:
  -17 (T): Major reorganization may be needed while on-line.
  -18 (C): Addition can be only "marked up" and then made definitive by batch.
  -19 (C): Common processing conceivable.
  -20 (C): Can be combined: Add = define null + change.
  -21 (T): Adding a record can imply adding others.
  -22 (T): See 21.
  -23 (T): Adding record can jeopardize integrity.
  -25 (T): Complicated on addition.
  -31 (T): Query language should support additions.
  -51 (T): Complicated on addition.
  -64 (T): If additions are only "marked up", maintenance can be made more time consuming.

-74 (T): Depending on how additions are handled response time can be degraded significantly.

-78 (C): Consistent with possible addition schemes.

-79 (C): Similar to (78).

* Requirement 15 is related to:

-19 (C): Common processing conceivable.

-21 (T): Made more complicated by (21).

-22 (T): Similar to (21).

-23 (T): Must be supported (explicit on change).

-51 (T): Complicated on change.

* Requirement 16 is related to:

-17 (T): Major reorganization may be needed while on-line.

-18 (C): Deletion can be only "marked up", then consolidated by batch.

-19 (C): Common processing possible.

-21 (T): Deleting record may imply deleting and/or changing others.

-22 (T): Similar to (21).

-23 (T): Deleting records may jeopardize integrity.

-25 (T): Complicated by deletion.

-31 (T): Expressable in query language.

-51 (T): Record may have to be restored.

-64 (T): Maintenance may have to take care of "half processed" deletion.

-74 (T): Depending on how deletions are processed response time can be degraded significantly.

-78 (C): Consistent with possible deletion procedures.

-79 (C): Similar to (78).

* Requirement 17 is related to:

-18 (C): Update can be only indicated and consolidated later.

-19 (C): Common processing conceivable.

-21 (T): Updating made more complex.

-22 (T): See (21).

-23 (T): Maintaining integrity a constraint.

-25 (T): Jeopardized.

-51 (T): Complicated on update.

-64 (T): Maintenance made more time consuming by certain possible updating procedures.

-78 (C): Consistent with possible updating procedures.

-79 (C): See (78).

* Requirement 18 is related to:

-78 (C): Common processing possible.

-79 (C): Common processing possible.

-87 (C): Batch maintenance may be cheaper.

* Requirement 19 is related to:

-21 (T): Bulk update to support it.

-22 (T): Similar to (21).

-23 (C): Easier on bulk update.

-25 (C): Keeping "old copy" may facilitate this.

-60 (C): Common processing conceivable.

-62 (T): Recovery may involve bulk update; time constraints.

-63 (T): Similar to (62).

* Requirement 20 is related to:

-31 (T): DDL able to specify this.

-71 (T): Compatible representations.

* Requirement 21 is related to:

-23 (T): Jeopardizes integrity.

-25 (T): Jeoppardized by (21).

-50 (C): May permit more explicit description of query status.

-51 (T): Requires keeping track of triggered computations.

- 76 (T): Triggered computations potential source of deadlocks.

* Requirement 22 is related to:

-23 (T): Jeopardizes integrity maintenence.

-25 (T): Made more difficult if (22) has to be supported.

-50 (C): May permit more explicit description of query status.

-51 (T): Requires keeping track of triggered computations.

-76 (T): Potential source of deadlocks.

* Requirement 23 is related to:
  -25 (C): Facilitated by (23).
  -26 (C): History of transactions may be used to restore integrity.
  -27 (C): Facilitates (23).
  -51 (C): Restoring previous integrity status made easier.
  -65 (C): Facilitated.
  -75 (T): More users make lockout processing trickier.
  -76 (T): Similar to (75).

* Requirement 24 is related to:
  -75 (T): See 23 above.
  -76 (T): See 23 above.

* Requirement 25 is related to:
  -51 (C): Compatible.
  -62 (T): Meet constraints.
  -63 (T): Meet constraints.
  -76 (C): May facilitate processing of deadlocks.

* Requirement 26 is related to:
  -51 (T): Cancelled queries require special processing.
  -68 (C): Common processing possible.

* Requirement 27 is related to:
  -28 (T): Support it.
  -29 (T): Support it.
  -30 (T): Support it.
  -56 (T): Support it.

* Requirement 31 isrelated to:
  -32 thru 40, 42,43,49,50,51,52,53,54 (T): Must be expressable.
  -73 (T): Bounds on time can put constraints on language characteristics (e.g., compile time

* Requirement 33 is related to:
  -36 (T): These relational conditionals must be supported.

* Requirement 37 is related to:
  -38 (C): Common processing (front end) possible.
  -52 (C): Similar to (38).

* Requirement 38 is related to:
  -39 (C): Enriches formatting capabilities.

* Requirement 39 is related to:
  -40 (C): Common processing conceivable.
  -42 (T): Must be one capability of (39).
  -44 (T): Similar to (42).
  -69 (C): Output procedures may be shared.

* Requirement 41 is related to:
  -51 (C): Common processing (front end) possible.
  -52 (C): Similar to (51).

* Requirements 45 thru 47 are related to:
  -77 (T): Terminal characteristics.
  -86 (T): Terminal costs.

* Requirement 48 is related to:
  -86 (T): Constraints on costs.

* Requirement 51 is related to:
  -68 (T): Cancelled requests must not be recorded.

* Requirement 52 is related to:
  -53 (C): Common processing conceivable.
  -54 (C): See 53.

* Requirement 53 is related to:
  -54 (C): Almost same processing except for final routing and control characters.

* Requirement 54 is related to:
  -55 (C): Messages can be treated as routed output.

* Requirement 56 is related to:
  -57 (C): Security control somewhat facilitated.
  -68 (C): Information can be gathered at signon time.

* Requirement 57 is related to:
  -58 (T): Master terminal can't be "hardwired".

* Requirement 59 is related to:
  -60 (C): Utilities can be tailored to meet constraints.

* Requirement 61 isrelated to:
  -86(T): Costs of reliable hardware.

* Requirement 62 is related to:
  -63 (T): Compatible constraints.
  -86 (T): See (61) above.
  -87 (C): Good maintenance can keep recovery time down.

* Requirement 63 is related to:
  -86 (T): Similar to (62).
  -87 (C): See (62).

* Requirement 64 is related to:
  -78 (C): Maintenance can take advantage of this.
  -82 (C): Common processing possible.
  -83 (C): See (82).
  -84 (C): Maintenance can be done through this.

* Requirement 65 is related to:
  -66 (C): Hardware characteristics.
  -67 (C): Possibility of "old version" disks.
  -86 (T): Hardware capabilities' costs.

* Requirement 66 is related to:
  -76 (C): Possibility of parallel processing.
  -86 (T): Hardware costs.

* Requirement 67 is related to:
  -70 (C): Transporting disks a possibility.
  -71 (T): Compatibility.
  -73 (T): Disks capable of required performance.
  -74 (T): Avoid scattering files in different disks.
  -76 (T): Concurrent queries may conflict as to needed disks.
  -81 (C): Facilitated.
  -82 (C): Added flexibility for tuning.
  -83 (C): Similar to (82).
  -84 (C): See (82).
  -86 (T): Disk units' costs.

* Requirement 68 is related to:
  -69 (T): Organize recording to facilitate reporting.

* Requirement 70 is related to:
  -71 (C): No need for data translation to transport application.
  -72 (C): Transport can be direct.
  -80 (T): Disk storage space enough for compatibility.
  -86 (T): Compatibility costs.

* Requirement 71 is related to:
  -72 (C): Transport can be made directly.

* Requirement 73 is related to:
  -74 (C): Compatible objectives.
  -t5 (T): Overhead for multi-user can jeopardize response time.
  -76 (T): Similar to (75).

* Requiremnt 74 is related to:
  -78 (C): Response time can be improved by reorganization upon file growth.
  -83 (C): (74) may be met by manipulating controls.

* Requirement 75 is related to:
  -77 (T): Less idle time for CPU due to fast output can imply degraded multi-user operation.

* Requirement 76 is related to:

  -77 (T): Similar to 75 above.

* Requirement 78 is related to:

  -79 (C): May make (79) more efficient and/or flexible.

  -84 (C): Supported by (78).

* Requirement 79 is related to:

  -80 (C): Released areas are reused.

* Requirement 81 is related to:

  - 82 (C): This is in fact implied by (81): making it explicit helps.

  -84 (C): Restructuring data may facilitate releasing storage space to be used growing files.

  -85 (T): Features called for by (81) may delay development.

* Requirement 82 is related to:

  -83 (C): Common processing possible.

  -85 (T): tuning facilities may have to be themselves tuned up.

  -86 (T): Added costs may be considerable.

APPENDIX D

SUMMARY OF ANALYSIS (*)

```
 REQ:
ENGR
:
*89/1;2,4,5,18,19,27,59,60/
:
2;5,7,8,12,14,15,16,22,23,25,29,33,51,74,77,79,83/
:
3;6,12,13,20,21,22,27,31/4;5/5;8,14,15,16,17,84,85/
:
6;20,21,22,23,74,75/7;10,19,23,27,29,77,82/
:
9;10,11,12,13/10;11,12,13/11;12,13/12;13/
:
13;15,20,21,22/
:
14;17,18,19,20,21,22,23,25,31,51,64,75,79,80/
:
15;19,21,22,23,25,51/
:
16;17,18,19,21,22,23,25,31,51,64,75,79,80/
:
17;18,19,21,22,23,25,51,64,79,80/18;79,80,89/
:
19;21,22,23,25,60,62,63/20;31,71/
:
21;23,26*5,50,51,77/22;23,25,50,51,77/
:
23;25,26,27,51,65,76,77/24;25,76,77/
:
25;51,62,63,77/26;51,68/27;28,29,30,56/
:
31;32,33,34,35,36,37,38,39,40,42,43,49,50,51,52,53,54,74/
:
33;36/37;38,52/38;39/39;40,42,44,69/41;51,52/45;78,88/48;88/
:
46;78,88/47;78,88/51;68/52;53,54/53;54/54;55/56;57,68/
:
57;58/59;60/61;88/62;63,88,89/63;88,89/64;79,83,84,85/
:
65;66,67,88/66;77,88/67;70,71,74,75,77,82,83,84,85,88/
:
68;69/70;71,73,81,88/71;72/**3/74;75,76,77/75;79,84/
:
76;78/77;78/79;80,85/80;81/82;83,85,86/83;84,86,88/
:
$

 NEW GRAPH ENTERED.

 REQ:
```

(*) The sequence of operations displayed here corresponds only approximately to the actual analytical sequence. Where not essential, some operations were ommitted.

```
NOLK

 RECORDED LINKS.
 FROM NODE     TO NODE(S):

      1   (  8)  2,  4,  5, 18, 19, 27, 59, 60,
      2   ( 18)  1,  5,  7,  8, 12, 14, 15, 16, 22, 23, 25, 29, 33, 51, 74, 77, 79, 83,
      3   (  8)  6, 12, 13, 20, 21, 22, 27, 31,
      4   (  2)  1,  5,
      5   ( 10)  1,  2,  4,  8, 14, 15, 16, 17, 84, 85,
      6   (  7)  3, 20, 21, 22, 23, 74, 75,
      7   (  8)  2, 10, 19, 23, 27, 29, 77, 82,
      8   (  2)  2,  5,
      9   (  4) 10, 11, 12, 13,
     10   (  5)  7,  9, 11, 12, 13,
     11   (  4)  9, 10, 12, 13,
     12   (  6)  2,  3,  9, 10, 11, 13,
     13   (  9)  3,  9, 10, 11, 12, 15, 20, 21, 22,
     14   ( 16)  2,  5, 17, 18, 19, 20, 21, 22, 23, 25, 31, 51, 64, 75, 79, 80,
     15   (  9)  2,  5, 13, 19, 21, 22, 23, 25, 51,
     16   ( 15)  2,  5, 17, 18, 19, 21, 22, 23, 25, 31, 51, 64, 75, 79, 80,
     17   ( 13)  5, 14, 16, 18, 19, 21, 22, 23, 25, 51, 64, 79, 80,
     18   (  7)  1, 14, 16, 17, 79, 80, 89,
     19   ( 13)  1,  7, 14, 15, 16, 17, 21, 22, 23, 25, 60, 62, 63,
     20   (  6)  3,  6, 13, 14, 31, 71,
     21   ( 13)  3,  6, 13, 14, 15, 16, 17, 19, 23, 25, 50, 51, 77,
     22   ( 14)  2,  3,  6, 13, 14, 15, 16, 17, 19, 23, 25, 50, 51, 77,
     23   ( 17)  2,  6,  7, 14, 15, 16, 17, 19, 21, 22, 25, 26, 27, 51, 65, 76, 77,
     24   (  3) 25, 76, 77,
     25   ( 14)  2, 14, 15, 16, 17, 19, 21, 22, 23, 24, 51, 62, 63, 77,
     26   (  3) 23, 51, 68,
     27   (  8)  1,  3,  7, 23, 28, 29, 30, 56,
     28   (  1) 27,
     29   (  3)  2,  7, 27,
     30   (  1) 27,
     31   ( 22)  3, 14, 16, 20, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 43, 49, 50, 51, 52, 53,
               54, 74,
```

```
32   (  1) 31,
33   (  3)  2, 31, 36,
34   (  1) 31,
35   (  1) 31,
36   (  2) 31, 33,
37   (  3) 31, 38, 52,
38   (  3) 31, 37, 39,
39   (  6) 31, 38, 40, 42, 44, 69,
40   (  2) 31, 39,
41   (  2) 51, 52,
42   (  2) 31, 39,
43   (  1) 31,
44   (  1) 39,
45   (  2) 78, 88,
46   (  2) 78, 88,
47   (  2) 78, 88,
48   (  1) 88,
49   (  1) 31,
50   (  3) 21, 22, 31,
51   ( 13)  2, 14, 15, 16, 17, 21, 22, 23, 25, 26, 31, 41, 68,
52   (  5) 31, 37, 41, 53, 54,
53   (  3) 31, 52, 54;
54   (  4) 31, 52, 53, 55,
55   (  1) 54,
56   (  3) 27, 57, 68,
57   (  2) 56, 58,
58   (  1) 57,
59   (  2)  1, 60,
60   (  3)  1, 19, 59,
61   (  1) 88,
62   (  5) 19, 25, 63, 88, 89,
63   (  5) 19, 25, 62, 88, 89,
64   (  7) 14, 16, 17, 79, 83, 84, 85,
65   (  4) 23, 66, 67, 88,
66   (  3) 65, 77, 88,
67   ( 11) 65, 70, 71, 74, 75, 77, 82, 83, 84, 85, 88,
68   (  4) 26, 51, 56, 69,
```

```
   69   (  2) 39, 68,
   70   (  5) 67, 71, 73, 81, 88,
   71   (  4) 20, 67, 70, 73,
   73   (  2) 70, 71,
   74   (  7)  2,  6, 31, 67, 75, 76, 77,
   75   (  7)  6, 14, 16, 67, 74, 79, 84,
   76   (  4) 23, 24, 74, 78,
   77   ( 11)  2,  7, 21, 22, 23, 24, 25, 66, 67, 74, 78,
   78   (  5) 45, 46, 47, 76, 77,
   79   (  9)  2, 14, 16, 17, 18, 64, 75, 80, 85,
   80   (  6) 14, 16, 17, 18, 79, 81,
   81   (  2) 70, 80,
   82   (  5)  7, 67, 83, 85, 86,
   83   (  7)  2, 64, 67, 82, 84, 86, 88,
   84   (  5)  5, 64, 67, 75, 83,
   85   (  5)  5, 64, 67, 79, 82,
   86   (  2) 82, 83,
   88   ( 12) 45, 46, 47, 48, 61, 62, 63, 65, 66, 67, 70, 83,
   89   (  3) 18, 62, 63,

(AVERAGE NO. OF LINKS PER NODE:     5.483).

 REQ:
ISOL

 ISOLATED NODES:
  72
  87
```

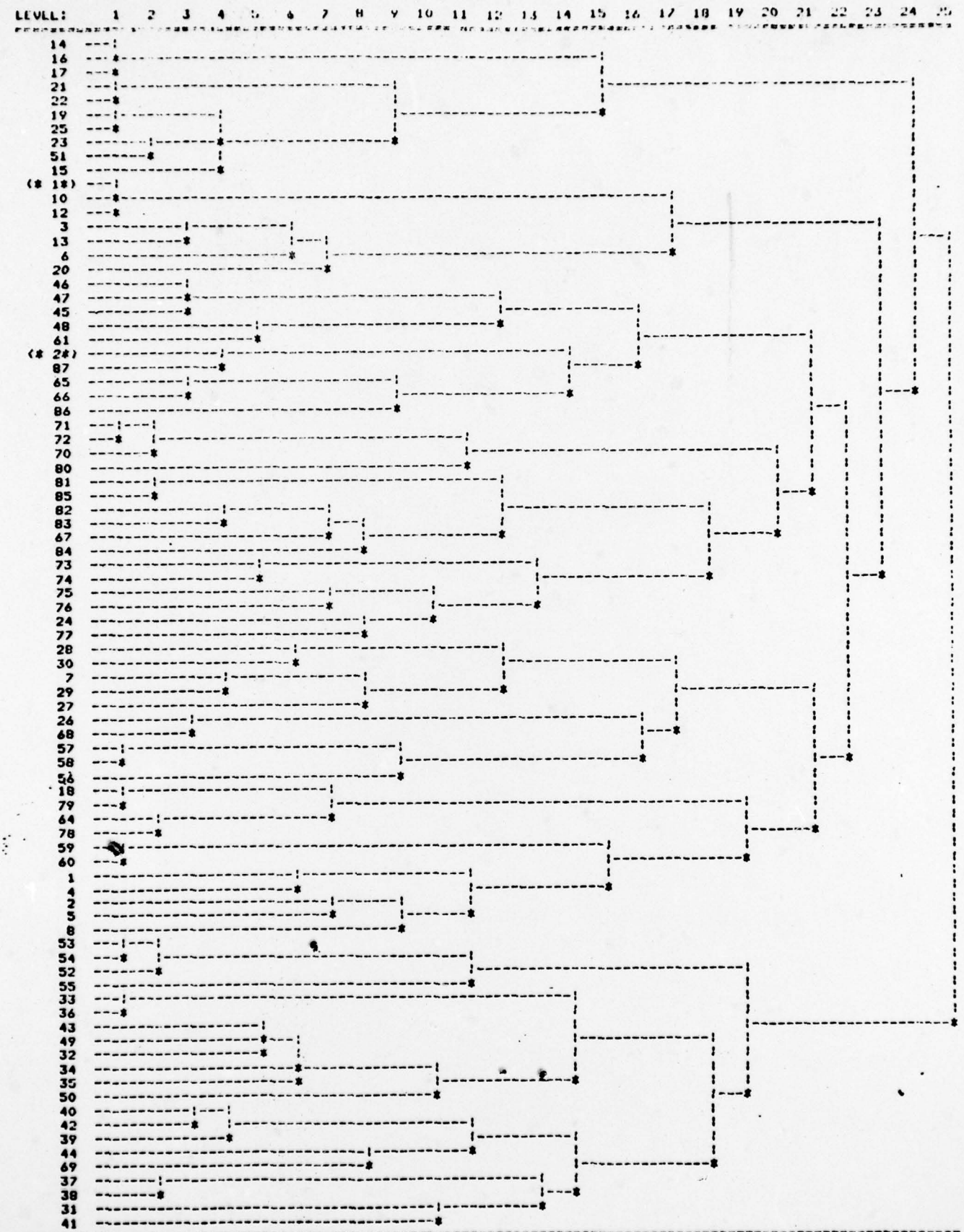
```
DENO
:
/72,87/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   72   87

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO. NEW NO.
 ---------------
    1        1
    2        2
    3        3
    4        4
    5        5
    6        6
    7        7
    8        8
    9        9
   10       10
        .
        .
        .
   62       62
   63       63
   64       64
   65       65
   66       66
   67       67
   68       68
   69       69
   70       70
   71       71
   73       72
   74       73
   75       74
   76       75
   77       76
   78       77
   79       78
   80       79
   81       80
   82       81
   83       82
   84       83
   85       84
   86       85
   88       86
   89       87

 REQ:
```

```
NOLK

 RECORDED LINKS.
 FROM NODE     TO NODE(S):

      1   (  8)  2,  4,  5, 18, 19, 27, 59, 60,
      2   ( 18)  1,  5,  7,  8, 12, 14, 15, 16, 22, 23, 25, 29, 33, 51, 73, 76, 78, 82,
      3   (  8)  6, 12, 13, 20, 21, 22, 27, 31,
      4   (  2)  1,  5,
      5   ( 10)  1,  2,  4,  8, 14, 15, 16, 17, 83, 84,
      6   (  7)  3, 20, 21, 22, 23, 73, 74,
      7   (  8)  2, 10, 19, 23, 27, 29, 76, 81,
      8   (  2)  2,  5,
      9   (  4) 10, 11, 12, 13,
     10   (  5)  7,  9, 11, 12, 13,
     11   (  4)  9, 10, 12, 13,
     12   (  6)  2,  3,  9, 10, 11, 13,
     13   (  9)  3,  9, 10, 11, 12, 15, 20, 21, 22,
     14   ( 16)  2,  5, 17, 18, 19, 20, 21, 22, 23, 25, 31, 51, 64, 74, 78, 79,
     15   (  9)  2,  5, 13, 19, 21, 22, 23, 25, 51,
     16   ( 15)  2,  5, 17, 18, 19, 21, 22, 23, 25, 31, 51, 64, 74, 78, 79,
     17   ( 13)  5, 14, 16, 18, 19, 21, 22, 23, 25, 51, 64, 78, 79,
     18   (  7)  1, 14, 16, 17, 78, 79, 87,
     19   ( 13)  1,  7, 14, 15, 16, 17, 21, 22, 23, 25, 60, 62, 63,
     20   (  6)  3,  6, 13, 14, 31, 71,
     21   ( 13)  3,  6, 13, 14, 15, 16, 17, 19, 23, 25, 50, 51, 76,
     22   ( 14)  2,  3,  6, 13, 14, 15, 16, 17, 19, 23, 25, 50, 51, 76,
     23   ( 17)  2,  6,  7, 14, 15, 16, 17, 19, 21, 22, 25, 26, 27, 51, 65, 75, 76,
     24   (  3) 25, 75, 76,
     25   ( 14)  2, 14, 15, 16, 17, 19, 21, 22, 23, 24, 51, 62, 63, 76,
     26   (  3) 23, 51, 68,
     27   (  8)  1,  3,  7, 23, 28, 29, 30, 56,
     28   (  1) 27,
     29   (  3)  2,  7, 27,
     30   (  1) 27,
     31   ( 22)  3, 14, 16, 20, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 43, 49, 50, 51, 52, 53,
               54, 73,
```

```
32    (  1) 31,
33    (  3)  2, 31, 36,
34    (  1) 31,
35    (  1) 31,
36    (  2) 31, 33,
37    (  3) 31, 38, 52,
38    (  3) 31, 37, 39,
39    (  6) 31, 38, 40, 42, 44, 69,
40    (  2) 31, 39,
41    (  2) 51, 52,
42    (  2) 31, 39,
43    (  1) 31,
44    (  1) 39,
45    (  2) 77, 86,
46    (  2) 77, 86,
47    (  2) 77, 86,
48    (  1) 86,
49    (  1) 31,
50    (  3) 21, 22, 31,
51    ( 13)  2, 14, 15, 16, 17, 21, 22, 23, 25, 26, 31, 41, 68,
52    (  5) 31, 37, 41, 53, 54,
53    (  3) 31, 52, 54,
54    (  4) 31, 52, 53, 55,
55    (  1) 54,
56    (  3) 27, 57, 68,
57    (  2) 56, 58,
58    (  1) 57,
59    (  2)  1, 60,
60    (  3)  1, 19, 59,
61    (  1) 86,
62    (  5) 19, 25, 63, 86, 87,
63    (  5) 19, 25, 62, 86, 87,
64    (  7) 14, 16, 17, 78, 82, 83, 84,
65    (  4) 23, 66, 67, 86,
66    (  3) 65, 76, 86,
67    ( 11) 65, 70, 71, 73, 74, 76, 81, 82, 83, 84, 86,
68    (  4) 26, 51, 56, 69,
```

```
    69   (  2) 39, 68,
    70   (  5) 67, 71, 72, 80, 86,
    71   (  4) 20, 67, 70, 72,
    72   (  2) 70, 71,
    73   (  7)  2,  6, 31, 67, 74, 75, 76,
    74   (  7)  6, 14, 16, 67, 73, 78, 83,
    75   (  4) 23, 24, 73, 77,
    76   ( 11)  2,  7, 21, 22, 23, 24, 25, 66, 67, 73, 77,
    77   (  5) 45, 46, 47, 75, 76,
    78   (  9)  2, 14, 16, 17, 18, 64, 74, 79, 84,
    79   (  6) 14, 16, 17, 18, 78, 80,
    80   (  2) 70, 79,
    81   (  5)  7, 67, 82, 84, 85,
    82   (  7)  2, 64, 67, 81, 83, 85, 86,
    83   (  5)  5, 64, 67, 74, 82,
    84   (  5)  5, 64, 67, 78, 81,
    85   (  2) 81, 82,
    86   ( 12) 45, 46, 47, 48, 61, 62, 63, 65, 66, 67, 70, 82,
    87   (  3) 18, 62, 63,

(AVERAGE NO. OF LINKS PER NODE:     5.609).

REQ:

 DIMN
  (PRECLUSTERING COMPLETE)

  PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
  CLUSTERS NOT TAKEN AS SINGLE NODES.
```

```
 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 1)  1
    2     ( 1)  2
    3     ( 1)  3
    4     ( 1)  4
    5     ( 1)  5
    6     ( 1)  6
    7     ( 1)  7
    8     ( 1)  8
    9     ( 2)  9 11
   10     ( 1) 10
   11     ( 1) 12
          .
          .
          .

   59     ( 1) 60
   60     ( 1) 61
   61     ( 2) 62 63
   62     ( 1) 64
   63     ( 1) 65
   64     ( 1) 66
   65     ( 1) 67
   66     ( 1) 68
   67     ( 1) 69
   68     ( 1) 70
   69     ( 1) 71
   70     ( 1) 72
   71     ( 1) 73
   72     ( 1) 74
   73     ( 1) 75
   74     ( 1) 76
   75     ( 1) 77
   76     ( 1) 78
   77     ( 1) 79
   78     ( 1) 80
   79     ( 1) 81
   80     ( 1) 82
   81     ( 1) 83
   82     ( 1) 84
   83     ( 1) 85
   84     ( 1) 86
   85     ( 1) 87

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.806
 DO YOU WANT TO PRINT THE TREE?
YES
 SET PAPER AND PRESS RETURN:
```



COLLAPSED OBJECTS:

(# 1#): 9 11
(# 2#): 62 63

```
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     (48)  1  2  4  5  7  8 18 24 26 27 28 29 30 45 46 47 48 56 57 58 59 60 61 62 63 64 65 66 67 68
              70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
    2     ( 8)  3  6  9 11 10 12 13 20
    3     (10) 14 15 16 17 19 21 22 23 25 51
    4     (21) 31 32 33 34 35 36 37 38 39 40 41 42 43 44 49 50 52 53 54 55 69


 REQ:
EVAL

 STRENGTH:  1.0691,
 COUPLING:  0.2632,
 MEASURE:   0.806.

 REQ:
DENO
:
/1,2,4,5,7,8,18,24,26,27,28,29,30,45,46,47,48,56,57,58,59,60,61,62,63,
:
64,65,66,67,68,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,
:
14,15,16,17,19,21,22,23,25,51,31,32,33,34,35,36,37,38,39,40,41,42,43,
:
44,49,50,52,53,54,55,69/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   1    2    4    5    7    8   14   15   16   17   18   19   21   22   23   24   25   26   27   28
  29   30   31   32   33   34   35   36   37   38   39   40   41   42   43   44   45   46   47   48
  49   50   51   52   53   54   55   56   57   58   59   60   61   62   63   64   65   66   67   68
  69   70   71   72   73   74   75   76   77   78   79   80   81   82   83   84   85   86   87
```

```
 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    3        1
    6        2
    9        3
   10        4
   11        5
   12        6
   13        7
   20        8

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:


HCM3
 BEST PARTITION MEASURE:    0.733
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 3)  1  2  8
    2     ( 5)  3  4  5  6  7

 REQ:
EVAL

 STRENGTH:  0.9333,
 COUPLING:  0.2000,
 MEASURE:   0.733.
```

```
 REQ:
DENO
:
/1,2,4,5,7,8,18,24,26,27,28,29,30,45,46,47,48,56,57,58,59,60,61,62,63,
:
64,65,66,67,68,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,
:
3,6,9,11,10,12,13,20,14,15,16,17,19,21,22,23,25,51/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
  21   22   23   24   25   26   27   28   29   30   45   46   47   48   51   56   57   58   59   60
  61   62   63   64   65   66   67   68   70   71   72   73   74   75   76   77   78   79   80   81
  82   83   84   85   86   87

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO. NEW NO.
 ---------------
   31        1
   32        2
   33        3
   34        4
   35        5
   36        6
   37        7
   38        8
   39        9
   40       10
   41       11
   42       12
   43       13
   44       14
   49       15
   50       16
   52       17
   53       18
   54       19
   55       20
   69       21
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.092
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    (16)  1  2  3  6  4  5  7  8  9 10 12 13 14 15 16 21
    2    ( 5) 11 17 18 19 20

 REQ:

          .

          .

          .


CLUSTER (NO) OBJECTS
--------------------
   1    (12)  1  2  4  5  7  8 27 28 29 30 59 60
   2    ( 4) 18 78 79 80
   3    ( 3) 24 75 76
   4    ( 5) 26 56 57 58 68
   5    ( 8) 45 46 47 48 61 65 66 77                          } MS 1
   6    (11) 64 67 73 74 81 82 83 84 85 86 87
   7    ( 3) 70 71 72
   8    ( 2) 62 63
   9    ( 3)  3  6 20                                         } MS 2
  10    ( 5)  9 10 11 12 13
  11    (10) 14 15 16 17 19 21 22 23 25 51                    MS 3
  12    (16) 31 32 33 34 35 36 37 38 39 40 42 43 44 49 50 69  } MS 4
  13    ( 5) 41 52 53 54 55

REQ:
```

```
PRLK

 LINKS BETWEEN CLUSTERS   1 &   2 :
                 1 -  18
                 2 -  78

 LINKS BETWEEN CLUSTERS   1 &   3 :
                 2 -  76
                 7 -  76

 LINKS BETWEEN CLUSTERS   1 &   4 :
                27 -  56

 LINKS BETWEEN CLUSTERS   1 &   5 :
 NONE.

 LINKS BETWEEN CLUSTERS   1 &   6 :
                 2 -  73
                 2 -  82
                 5 -  83
                 5 -  84
                 7 -  81

 LINKS BETWEEN CLUSTERS   1 &   7 :
 NONE.

 LINKS BETWEEN CLUSTERS   1 &   8 :
 NONE.

 LINKS BETWEEN CLUSTERS   1 &   9 :
                27 -   3

 LINKS BETWEEN CLUSTERS   1 &  10 :
                 2 -  12
                 7 -  10

 LINKS BETWEEN CLUSTERS   1 &  11 :
                 1 -  19
                 2 -  14
                 2 -  15
                 2 -  16
                 2 -  22
                 2 -  23
                 2 -  25
                 2 -  51
                 5 -  14
                 5 -  15
                 5 -  16
                 5 -  17
                 7 -  19
                 7 -  23
                27 -  23
                60 -  19
```

LINKS BETWEEN CLUSTERS 1 & 12 :
2 - 33

LINKS BETWEEN CLUSTERS 1 & 13 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 3 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 4 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 5 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 6 :
18 - 87
78 - 64
78 - 74
78 - 84

LINKS BETWEEN CLUSTERS 2 & 7 :
80 - 70

LINKS BETWEEN CLUSTERS 2 & 8 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 9 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 10 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 11 :
18 - 14
18 - 16
18 - 17
78 - 14
78 - 16
78 - 17
79 - 14
79 - 16
79 - 17

LINKS BETWEEN CLUSTERS 2 & 12 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 13 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 4 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 5 :
75 - 77
76 - 66
76 - 77

LINKS BETWEEN CLUSTERS 3 & 6 :
75 - 73
76 - 67
76 - 73

LINKS BETWEEN CLUSTERS 3 & 7 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 8 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 9 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 10 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 11 :
24 - 25
75 - 23
76 - 21
76 - 22
76 - 23
76 - 25

LINKS BETWEEN CLUSTERS 3 & 12 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 13 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 5 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 6 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 7 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 8 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 9 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 10 :
NONE.

```
LINKS BETWEEN CLUSTERS    4 &  11 :
                26 -  23
                26 -  51
                68 -  51

LINKS BETWEEN CLUSTERS    4 &  12 :
                68 -  69

LINKS BETWEEN CLUSTERS    4 &  13 :
NONE.

LINKS BETWEEN CLUSTERS    5 &   6 :
                45 -  86
                46 -  86
                47 -  86
                48 -  86
                61 -  86
                65 -  67
                65 -  86
                66 -  86

LINKS BETWEEN CLUSTERS    5 &   7 :
NONE.

LINKS BETWEEN CLUSTERS    5 &   8 :
NONE.

LINKS BETWEEN CLUSTERS    5 &   9 :
NONE.
```

.

.

.

APPENDIX E

We show below that unless two subgraphs are disjoint, the coupling between them increases if they are further decomposed.

Let:

- A, B be the two subgraphs;
- n_A, n_B (> 1) be the number of components in an arbitrary partition of A and B, i.e.:

$$\left.\begin{aligned} A &= \bigcup_{i=1}^{n_A} A_i \quad ; \quad \bigcap_{i\neq j} A_i,\ A_j = \emptyset, \quad i,j=1,\ldots,n_A \\ B &= \bigcup_{j=1}^{n_B} B_j \quad ; \quad \bigcap_{i\neq j} B_i,\ B_j = \emptyset, \quad i,j=1,\ldots,n_B \end{aligned}\right\} \quad \ldots\ldots (1)$$

- $L_{A,B}$ be the number of links between A and B; and
- L_{A_i,B_j} be the number of links between A_i and B_j.

Since $A_i \subset A$, $i=1,\ldots,n_A$, obviously

$$|A_i| < |A| \ , \ i=1,\ldots,n_A \qquad \ldots\ldots\ldots\ldots\ldots\ldots (2)$$

Similarly,

$$|B_j| < |B| \ , \ j=1,\ldots,n_B \qquad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

Furthermore, by (1) --i.e., because the components in a partition are mutually disjoint and span the complete graphs A and B--, it is true that

$$L_{A,B} = \sum_{\substack{i=1 \\ j=1}}^{\substack{n_B \\ n_A}} L_{A_i,B_j} .$$

By definition of coupling,

$$C(A, B) = \frac{L_{A,B}}{|A| \cdot |B|} = \frac{\sum_{\substack{i=1 \\ j=1}}^{\substack{n_B \\ n_A}} L_{A_i,B_j}}{|A| \cdot |B|} ,$$

and

$$C(A_i, B_j) = \frac{L_{A_i,B_j}}{|A_i| \cdot |B_j|} ,$$

so that, by (2) and (3),

$$C(A, B) < \sum_{\substack{i=1 \\ j=1}}^{\substack{n_B \\ n_A}} C(A_i, B_j) \qquad (\text{unless } L_{A,B} = 0), \text{Q.E.D.}$$